



Anexo 2

Pruebas para la obtención de títulos de Técnico y Técnico Superior

Convocatoria correspondiente al curso académico 2023-2024

(Resolución de 29 de diciembre de 2023 de la Dirección General de Educación Secundaria, Formación Profesional y Régimen Especial)

DATOS DEL ASPIRANTE			FIRMA
Apellidos:			
Nombre:	D.N.I. N.I.E. o Pasaporte:	Fecha:	

Código del ciclo:	Denominación completa del título:
IFCS02	Técnico Superior en Desarrollo de Aplicaciones Multiplataforma
Clave o código del módulo:	Denominación completa del módulo profesional:
490	Programación de Servicios y Procesos

INSTRUCCIONES GENERALES PARA LA REALIZACIÓN DE LA PRUEBA
<ul style="list-style-type: none">El examen tendrá una duración de 2 horas.La prueba consta de un examen tipo test con cuatro opciones de las cuales solamente una es correcta.Cada pregunta se responderá en el espacio dejado al efecto en la hoja de respuestas. Se rellenarán los recuadros para señalar la respuesta seleccionada.Si se quiere rectificar una respuesta contestada, se borrará toda la casilla de la respuesta incorrecta con Tipp-Ex o corrector, tal y como se puede apreciar en el siguiente ejemplo:<ul style="list-style-type: none">Se elimina la selección de b para seleccionar la opción d: <input type="checkbox"/>a <input type="checkbox"/>b <input type="checkbox"/>c <input checked="" type="checkbox"/>dSe dispondrá de una hoja para borrador (o de varias si se requieren), que será proporcionada por el centro. Esa hoja se entregará obligatoriamente al final junto con el examen, si bien nada de lo escrito en la hoja de borrador se valorará en la corrección.Sólo se utilizará bolígrafo negro o azul y Tipp-Ex o corrector, no permitiéndose usar bolígrafo rojo, lapicero, etcétera.No se podrá emplear ningún dispositivo electrónico.Cualquier tachadura o borrón en una respuesta podrá invalidar toda la puntuación de la pregunta asociada.

CRITERIOS DE CALIFICACIÓN Y VALORACIÓN
<ul style="list-style-type: none">El test se calificará sobre 10 puntos. Todas las preguntas se calificarán equitativamente con la misma cantidad de puntos. En cada pregunta se plantearán varias respuestas, y se deberá señalar la única que se considere correcta, según el caso. Cada respuesta correcta que se marque se valorará con 0,25 puntos, y si se marca alguna incorrecta, se valorará con una cantidad negativa equivalente a 0,075 puntos, es decir, se descontarán 0,075 puntos. Si no se está seguro de si una respuesta es correcta o no, y no se marca, no sumará ni restará puntos.Calificación final del módulo profesional:<ul style="list-style-type: none">El alumno obtendrá en el módulo profesional una calificación entera entre 1 y 10.Dicha calificación se calculará:<ul style="list-style-type: none">Si la calificación conseguida en la prueba es inferior a 5: se truncará dicha calificación.Si la calificación conseguida en la prueba es igual o superior a 5 y los decimales:<ul style="list-style-type: none">Son inferiores a 0'5: se redondeará al entero inferior más próximo.Son iguales o superiores a 0'5: se redondeará al entero superior más cercano.La anterior regla tiene una excepción: las notas de examen inferiores a 1 se redondearán a 1.



CONTENIDO DE LA PRUEBA

Parte 1: programación multiproceso. Resultados de aprendizaje: Desarrolla aplicaciones compuestas por varios procesos reconociendo y aplicando principios de programación paralela.

Tenemos el siguiente programa

```
public class ConteoMultiproceso {  
    public static void main(String[] args) {  
        Thread ascendente = new Thread() -> {  
            for (int i = 1; i <= 5; i++) {  
                System.out.println("Ascendente: " + i);  
                try {  
                    Thread.sleep(1000);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
                }  
            }  
        };  
  
        Thread descendente = new Thread() -> {  
            for (int i = 5; i > 0; i--) {  
                System.out.println("Descendente: " + i);  
                try {  
                    Thread.sleep(800);  
                } catch (InterruptedException e) {  
                    e.printStackTrace();  
                }  
            }  
        };  
  
        ascendente.start();  
        descendente.start();  
    }  
}
```

- 1) Si se modifica el intervalo de sueño del hilo ascendente de 1000 milisegundos a 500 milisegundos, ¿qué impacto tiene esto en la salida del programa?
 - a. El orden de salida de los números se invertirá completamente.
 - b. Los números del hilo ascendente se imprimirán más rápidamente, resultando en una terminación anticipada comparada con el hilo descendente.
 - c. No afectará el orden en que se imprimen los números.
 - d. Causará que ambos hilos terminen al mismo tiempo.
- 2) Considerando las buenas prácticas en programación multihilo, ¿cuál sería una crítica válida de este código?
 - a. El uso de System.out.println es ineficiente y debe evitarse en ambientes multihilo.



- b. La manipulación directa de hilos sin utilizar un ejecutor o un servicio de hilos es una práctica desaconsejada.
 - c. Los intervalos de sueño deben ser idénticos para evitar condiciones de carrera.
 - d. No hay ninguna mejora necesaria; el código sigue todas las buenas prácticas de multihilo.
- 3) Si el hilo ascendente se inicia primero, ¿qué secuencia de números es correcta según la salida esperada del programa?**
- a. 1, 2, 3, 4, 5 seguido inmediatamente por 5, 4, 3, 2, 1
 - b. Una mezcla de números en orden ascendente y descendente debido a la programación multiproceso.
 - c. 5, 4, 3, 2, 1 seguido inmediatamente por 1, 2, 3, 4, 5
 - d. Sólo números en orden ascendente.
- 4) ¿Qué excepción manejan los bloques try-catch en este programa?**
- a. IOException
 - b. NumberFormatException
 - c. InterruptedException
 - d. ArithmeticException
- 5) Considerando la programación multiproceso en Java, ¿qué afirmación es verdadera sobre la ejecución del programa proporcionado?**
- a. Los hilos ascendente y descendente se ejecutan secuencialmente.
 - b. El hilo ascendente siempre termina antes que el hilo descendente.
 - c. La salida del programa puede variar en diferentes ejecuciones.
 - d. Los números siempre se imprimen en orden ascendente seguido por orden descendente sin intercalarse.
- 6) ¿Cuál es el efecto de las diferencias en los intervalos de tiempo de sleep() entre los hilos ascendente y descendente en la salida esperada del programa ConteoMultiproceso?**
- a. No tiene ningún efecto; los números siempre se imprimen en el mismo orden porque los hilos se inician en secuencia.
 - b. Hace que el hilo descendente complete su ejecución antes del hilo ascendente, lo que podría causar que los números en orden descendente se impriman más frecuentemente al principio.
 - c. Hace que el hilo ascendente espere a que el hilo descendente termine antes de imprimir su siguiente número, garantizando así una salida ordenada.
 - d. Aumenta la probabilidad de intercalación de los números impresos en orden ascendente y descendente, debido a la diferencia en los tiempos de pausa entre iteraciones de los bucles en cada hilo.
- 7) En el contexto de la programación concurrente en Java, ¿cuál es la principal diferencia entre el uso de synchronized y ReentrantLock para controlar el acceso a un recurso compartido?**
- a. ReentrantLock permite un control más fino sobre las políticas de bloqueo y puede mejorar el rendimiento en ciertos escenarios, mientras que synchronized es más fácil de implementar pero menos flexible.
 - b. synchronized es capaz de bloquear automáticamente los recursos sin necesidad de desbloquearlos, mientras que ReentrantLock requiere que el programador se encargue manualmente de su bloqueo y desbloqueo.



Unión Europea

Fondo Social Europeo
"El FSE invierte en tu futuro"



Comunidad de Madrid

- c. ReentrantLock solo se puede utilizar en código que no se ejecuta en un entorno multihilo, mientras que synchronized no tiene tal restricción.
- d. No hay diferencias significativas; ReentrantLock es simplemente una nueva implementación de synchronized introducida para compatibilidad con versiones anteriores de Java.

8) ¿Qué estrategia de programación multiproceso evita eficazmente el problema de la inanición en sistemas concurrentes?

- a. Priorizar los hilos según su tiempo de espera, asegurando que todos los hilos eventualmente obtengan acceso a los recursos (algoritmo de envejecimiento).
- b. Utilizar un esquema de prioridades fijas, donde los hilos con mayor prioridad siempre se ejecutan antes que aquellos con menor prioridad.
- c. Asignar todos los recursos al hilo que primero los solicite, independientemente de su carga de trabajo o prioridad.
- d. Eliminar cualquier forma de sincronización, permitiendo que todos los hilos compitan libremente por los recursos.

Parte 2: programación multihilo. Resultados de aprendizaje: Desarrolla aplicaciones compuestas por varios hilos reconociendo y aplicando principios de programación paralela.

Tenemos el siguiente programa:



Unión Europea

Fondo Social Europeo
"El FSE invierte en tu futuro"



Comunidad de Madrid

```
public class ProductorConsumidor {  
    public static void main(String[] args) {  
        BufferCompartido buffer = new BufferCompartido(10);  
  
        Thread productor = new Thread(new Productor(buffer));  
        Thread consumidor = new Thread(new Consumidor(buffer));  
        productor.setPriority(Thread.MAX_PRIORITY);  
        consumidor.setPriority(Thread.MIN_PRIORITY);  
  
        productor.start();  
        consumidor.start();  
  
        Thread monitor = new Thread(new Monitor(buffer));  
        monitor.setDaemon(true);  
        monitor.start();  
    }  
}
```

```
class BufferCompartido {  
    private Queue<Integer> queue;  
    private int capacidad;  
  
    public BufferCompartido(int capacidad) {  
        this.capacidad = capacidad;  
        this.queue = new LinkedList<>();  
    }  
  
    public synchronized void agregar(int valor) {  
        while (queue.size() == capacidad) {  
            try {  
                wait();  
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        }  
        queue.add(valor);  
        notifyAll();  
    }  
}
```



Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"



IES Rey Fernando VI



Comunidad de Madrid

```
public synchronized int obtener() {  
    while (queue.isEmpty()) {  
        try {  
            wait();  
        } catch (InterruptedException e) {  
            Thread.currentThread().interrupt();  
        }  
    }  
    int valor = queue.poll();  
    notifyAll();  
    return valor;  
}  
}
```

```
class Productor implements Runnable {  
    private BufferCompartido buffer;  
  
    public Productor(BufferCompartido buffer) {  
        this.buffer = buffer;  
    }  
  
    public void run() {  
        for (int i = 0; i < 50; i++) {  
            buffer.agregar(i);  
            System.out.println("Producido: " + i);  
        }  
    }  
}
```

```
class Consumidor implements Runnable {  
    private BufferCompartido buffer;  
  
    public Consumidor(BufferCompartido buffer) {  
        this.buffer = buffer;  
    }  
  
    public void run() {  
        for (int i = 0; i < 50; i++) {  
            System.out.println("Consumido: " + buffer.obtener());  
        }  
    }  
}
```



```
class Monitor implements Runnable {  
    private BufferCompartido buffer;  
  
    public Monitor(BufferCompartido buffer) {  
        this.buffer = buffer;  
    }  
  
    public void run() {  
        while (true) {  
            System.out.println("Elementos en el buffer: " + buffer.queue.size());  
            try {  
                Thread.sleep(1000);  
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        }  
    }  
}
```

9) Dado el uso de `synchronized` en el método `incrementarContador`, ¿qué garantiza este modificador?

- a. Que cualquier hilo puede acceder al método si tiene suficiente prioridad.
- b. Que solo un hilo a la vez dentro del mismo objeto puede ejecutar cualquier bloque `synchronized`.
- c. Que solo un hilo a la vez, independientemente del objeto, puede ejecutar `incrementarContador`.
- d. Que el contador se incrementará sin ninguna pausa entre las operaciones.

10) ¿Qué implica la llamada a `Thread.sleep(100)` dentro de `incrementarContador`?

- a. Que el hilo actual espera exactamente 100 milisegundos antes de continuar.
- b. Que todos los hilos excepto el actual se pausan por 100 milisegundos.
- c. Que el hilo actual libera el bloqueo sobre el objeto contador mientras duerme.
- d. Que el hilo actual intenta ceder su tiempo restante de ejecución a otros hilos.

11) Considerando la implementación del método `incrementarContador`, ¿cuál de los siguientes aspectos NO es manejado directamente por el código proporcionado?

- a. Prioridad de los hilos.
- b. Manejo de excepciones durante la espera (`Thread.sleep`).
- c. Inanición (starvation) de hilos.
- d. Comunicación directa entre `hilo1` y `hilo2`.

12) Dado que `incrementarContador` es un método `synchronized`, ¿cuál de las siguientes afirmaciones es verdadera cuando un hilo está ejecutando este método?

- a. Ningún otro hilo puede ejecutar métodos `synchronized` del mismo objeto o clase.
- b. Otros hilos pueden ejecutar métodos `synchronized` de la misma instancia sin restricciones.
- c. Solo hilos con mayor prioridad pueden interrumpir y ejecutar el método `incrementarContador`.
- d. Hilos de otras instancias no pueden ejecutar métodos `synchronized` estáticos de la misma clase.



13) ¿Qué método de la API de Java permite a un hilo ceder deliberadamente su tiempo de procesamiento actual a otros hilos?

- a. Thread.yield()
- b. Thread.sleep(long millis)
- c. Thread.wait()
- d. Thread.join()

14) En el contexto de hilos demonio en Java, ¿cuál de las siguientes afirmaciones describe adecuadamente su comportamiento?

- a. Los hilos demonio tienen la capacidad de prevenir que la JVM se termine.
- b. Un hilo demonio se ejecuta en un plano de mayor prioridad en comparación con los hilos de usuario.
- c. La JVM se termina cuando solo quedan hilos demonio ejecutándose.
- d. Los hilos demonio no pueden acceder a recursos del sistema como archivos y redes.

15) Considerando la gestión avanzada de hilos, ¿qué estrategia se recomienda para manejar múltiples hilos que acceden a un recurso compartido para evitar el bloqueo y la espera activa?

- a. Uso exclusivo de variables volatile.
- b. Aplicar Thread.sleep() frecuentemente para dar tiempo a otros hilos.
- c. Implementar un mecanismo de bloqueo fino usando Lock y condiciones de java.util.concurrent.locks.
- d. Incrementar la prioridad de los hilos que acceden más frecuentemente al recurso compartido.

16) Dado el contexto del programa multihilo proporcionado y considerando las características avanzadas de concurrencia en Java, supongamos que se introduce una nueva variable static int lecturasContador = 0 que se incrementa cada vez que un hilo lee el valor de contador después de incrementarlo.

Se implementa un nuevo método synchronized static void leerIncrementarContador() que realiza esta operación, y se ajusta el programa para utilizar este método.

Con estas modificaciones, ¿cuál de las siguientes afirmaciones describe adecuadamente el impacto en el comportamiento del programa y la concurrencia, asumiendo que se usa correctamente el patrón de bloqueo de doble comprobación (double-checked locking) para optimizar el acceso al método leerIncrementarContador?

- a. El uso del patrón de bloqueo de doble comprobación elimina la necesidad de sincronizar leerIncrementarContador, permitiendo que múltiples hilos actualicen lecturasContador y contador simultáneamente sin causar condiciones de carrera.
- b. A pesar del patrón de bloqueo de doble comprobación, la sincronización del método leerIncrementarContador sigue siendo necesaria para garantizar la atomicidad de las operaciones de lectura y escritura en contador y lecturasContador, manteniendo la consistencia de los datos bajo concurrencia.
- c. La implementación del patrón de bloqueo de doble comprobación reduce significativamente el rendimiento del programa debido a la sobrecarga de verificar repetidamente el estado de contador antes de adquirir el bloqueo.



- d. Utilizar el patrón de bloqueo de doble comprobación permite que leerIncrementarContador ejecute operaciones de lectura sin sincronización, pero las operaciones de escritura en contador y lecturasContador deben permanecer dentro de un bloque sincronizado para prevenir condiciones de carrera.

Parte 3: programación de comunicación en red. Resultados de aprendizaje: Programa mecanismos de comunicación en red empleando sockets y analizando el escenario de ejecución.

Tenemos Servidor TCP:

```
public class TCPServer {  
    public static void main(String args[]) throws IOException {  
        ServerSocket serverSocket = new ServerSocket(6789);  
        System.out.println("Servidor iniciado...");  
  
        while(true) {  
            Socket connectionSocket = serverSocket.accept();  
            new ClientHandler(connectionSocket).start();  
        }  
    }  
}  
  
class ClientHandler extends Thread {  
    private Socket connectionSocket;  
  
    public ClientHandler(Socket connectionSocket) {  
        this.connectionSocket = connectionSocket;  
    }  
  
    public void run() {  
        try {  
            BufferedReader inFromClient = new BufferedReader(new  
InputStreamReader(connectionSocket.getInputStream()));  
            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());  
  
            String clientSentence = inFromClient.readLine();  
            System.out.println("Recibido: " + clientSentence);  
            outToClient.writeBytes(clientSentence.toUpperCase() + '\n');  
        } catch (IOException e) {  
            e.printStackTrace();  
        }  
    }  
}
```



Cliente TCP:

```
public class TCPClient {  
    public static void main(String argv[]) throws Exception {  
        String sentence;  
        String modifiedSentence;  
  
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));  
        Socket clientSocket = new Socket("localhost", 6789);  
  
        DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());  
        BufferedReader inFromServer = new BufferedReader(new  
        InputStreamReader(clientSocket.getInputStream()));  
  
        sentence = inFromUser.readLine();  
        outToServer.writeBytes(sentence + '\n');  
  
        modifiedSentence = inFromServer.readLine();  
        System.out.println("DEL SERVIDOR: " + modifiedSentence);  
  
        clientSocket.close();  
    }  
}
```

17) ¿Qué método se utiliza para enviar datos al cliente en el programa servidor?

- a. writeBytes()
- b. send()
- c. write()
- d. flush()

18) ¿Cómo captura el servidor las entradas del cliente?

- a. Utilizando un DataInputStream directamente desde el Socket.
- b. Mediante un BufferedReader envolviendo un InputStreamReader, que a su vez lee desde el InputStream del socket.
- c. Leyendo directamente desde el InputStream del Socket.
- d. Usando un Scanner con el InputStream del socket.

19) ¿Qué permite la clase ClientHandler en el servidor?

- a. Ejecutar el servidor en un solo hilo para simplificar la gestión de conexiones.
- b. Aceptar múltiples conexiones de clientes sin bloquear el hilo principal del servidor.
- c. Gestionar únicamente la entrada y salida de datos sin manejar conexiones concurrentes.
- d. Limitar el número de clientes que pueden conectarse al servidor simultáneamente.

20) ¿Qué ocurre si el método readLine() del objeto BufferedReader inFromClient devuelve null?

- a. El servidor enviará una cadena "null" al cliente.
- b. El servidor terminará abruptamente con una excepción.
- c. Significa que el cliente cerró su socket.



- d. El servidor reinicia la conexión con el cliente.

21) En el contexto de este servidor TCP, ¿qué sucedería si el método run() de ClientHandler lanza una excepción no capturada?

- a. El servidor se detendría por completo.
- b. El hilo en el que ocurrió la excepción terminaría, pero el servidor continuaría aceptando nuevas conexiones.
- c. El cliente se reconectaría automáticamente.
- d. Se crearía un nuevo hilo para reemplazar al que falló.

22) ¿Qué característica de TCP garantiza que los datos enviados desde el cliente lleguen en el mismo orden al servidor?

- a. Multiplexación
- b. Control de flujo
- c. Control de congestión
- d. Control de secuencia

23) ¿Qué implica el método accept() bloqueante en el servidor TCP respecto al modelo de programación utilizado?

- a. El servidor no puede aceptar nuevas conexiones hasta que no se procese completamente la actual.
- b. El servidor utiliza un modelo de programación basado en eventos para gestionar las conexiones.
- c. Cada conexión entrante se maneja en su propio hilo, permitiendo que accept() bloquee sin afectar la capacidad del servidor para procesar otras conexiones simultáneamente.
- d. accept() crea un nuevo hilo para cada conexión, evitando que el servidor se bloquee al esperar conexiones.

24) Considerando el manejo de errores en la comunicación de red, ¿qué mejoraría la robustez del servidor TCP presentado?

- a. Implementar un mecanismo de timeout para las conexiones.
- b. Usar un ExecutorService para limitar el número de hilos simultáneos y gestionarlos de manera más eficiente.
- c. Cerrar el ServerSocket después de aceptar una conexión para evitar sobrecargas.
- d. Reemplazar BufferedReader y DataOutputStream por Scanner

Parte 4: Generación de servicios web. Resultados de aprendizaje: Desarrolla aplicaciones que ofrecen servicios en red, utilizando librerías de clases y aplicando criterios de eficiencia y disponibilidad.

25) ¿Cuál de las siguientes afirmaciones es verdadera respecto al uso de librerías de clases en la programación de redes?

- a. Solo proporcionan estructuras de datos para almacenar información.
- b. Permiten el manejo de excepciones específicas de la red sin soporte para la programación asíncrona.
- c. Facilitan la creación de aplicaciones cliente-servidor mediante la encapsulación de protocolos estándar.
- d. Son exclusivas para la programación en Java.

26) ¿Qué método se utiliza típicamente para finalizar una conexión TCP de manera segura en una aplicación de red?

- a. Envío de un paquete con la bandera FIN establecida.



- b. Cierre del socket sin enviar ningún dato adicional.
 - c. Envío de un mensaje de aplicación indicando el cierre.
 - d. Reinicio de la conexión mediante el envío de un paquete con la bandera RST.
- 27) En el contexto de la transmisión de datos utilizando TCP, ¿qué mecanismo asegura que los datos lleguen en orden y sin errores al receptor?**
- a. Control de flujo
 - b. Multiplexación
 - c. Control de congestión
 - d. Numeración de secuencia y acuse de recibo
- 28) Cuando se desarrolla una aplicación cliente que utiliza HTTP para solicitar recursos web, ¿qué componente es esencial para interpretar las respuestas del servidor?**
- a. Un analizador de URL
 - b. Un intérprete de JavaScript
 - c. Un analizador de HTML
 - d. Un gestor de cookies
- 29) Considerando una aplicación que maneja múltiples conexiones cliente simultáneamente, ¿cuál de las siguientes técnicas se considera más eficiente para gestionar la concurrencia?**
- a. Multi-threading
 - b. Polling de eventos
 - c. Multiplexación de E/S
 - d. Ejecución secuencial con tiempos de espera
- 30) ¿Qué modelo de programación facilita el escalado de servicios de red para manejar un alto volumen de solicitudes sin aumentar linealmente los recursos del sistema?**
- a. Programación orientada a objetos
 - b. Programación basada en eventos
 - c. Programación procedimental
 - d. Programación funcional
- 31) En el desarrollo de aplicaciones de red, la utilización de objetos predefinidos (como sockets) es común. ¿Qué característica de los sockets TCP predefinidos garantiza la entrega confiable de paquetes?**
- a. Soporte para la transmisión en modo datagrama
 - b. Control de flujo y corrección de errores
 - c. Integración con sistemas de cifrado de datos
 - d. Compatibilidad con protocolos de capa de aplicación
- 32) En el contexto de asegurar la comunicación de datos en aplicaciones de red, especialmente al implementar transacciones sensibles, ¿cuál de las siguientes opciones representa mejor una estrategia avanzada para la seguridad de datos transmitidos usando protocolos estándar como HTTP, SMTP, y FTP?**
- a. Encriptación a nivel de aplicación utilizando protocolos específicos como HTTPS, S/MIME para SMTP, y FTPS para FTP, asegurando que los datos sean ilegibles durante su tránsito por la red.
 - b. Implementación de firewalls de red y sistemas de detección de intrusiones (IDS) para monitorizar y bloquear tráfico sospechoso, asegurando que solo las solicitudes legítimas sean procesadas.



Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"



IES Rey Fernando VI



Comunidad de Madrid

- c. Uso de técnicas de compresión de datos para reducir el tamaño de los paquetes transmitidos, disminuyendo así la ventana de oportunidad para ataques de interceptación de datos.
- d. Establecimiento de una red privada virtual (VPN) entre el cliente y el servidor, proporcionando una capa adicional de seguridad para el tráfico de datos que transita por redes públicas.

Parte 5: Utilización de técnicas de programación segura. Resultados de aprendizaje: Protege las aplicaciones y los datos definiendo y aplicando criterio de seguridad en el acceso, almacenamiento y transmisión de la información.

Tenemos el siguiente programa:



```
public class SecureServer {

    private static final int PORT = 8443;
    private static final String KEYSTORE_PATH = "keystore.jks";
    private static final String KEYSTORE_PASSWORD = "changeit";

    public static void main(String[] args) {
        try {
            KeyStore keyStore = KeyStore.getInstance("JKS");
            try (InputStream keyStoreData = new FileInputStream(KEYSTORE_PATH)) {
                keyStore.load(keyStoreData, KEYSTORE_PASSWORD.toCharArray());
            }

            KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance("SunX509");
            keyManagerFactory.init(keyStore, KEYSTORE_PASSWORD.toCharArray());

            SSLContext sslContext = SSLContext.getInstance("TLS");
            sslContext.init(keyManagerFactory.getKeyManagers(), null, null);

            SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();
            SSLServerSocket sslServerSocket = (SSLServerSocket)
            sslServerSocketFactory.createServerSocket(PORT);

            System.out.println("Servidor seguro escuchando en el puerto " + PORT);

            try (SSLSocket sslSocket = (SSLSocket) sslServerSocket.accept()) {
                // Comunicación segura establecida
                InputStream input = sslSocket.getInputStream();
                BufferedReader reader = new BufferedReader(new InputStreamReader(input));
                String line = reader.readLine(); // Leer datos enviados por el cliente
                System.out.println("Cliente dice: " + line);
                // Responder al cliente
                OutputStream output = sslSocket.getOutputStream();
                PrintWriter writer = new PrintWriter(output, true);
                writer.println("Hola, cliente seguro!");
            }

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```



33) ¿Qué almacén de claves utiliza el programa SecureServer?

- a. trustStore.jks
- b. keystore.jks
- c. clientStore.jks
- d. serverStore.jks

34) ¿Qué protocolo de seguridad implementa el servidor para las conexiones?

- a. HTTP
- b. HTTPS
- c. SSL
- d. TLS

35) ¿Cuál de las siguientes afirmaciones es cierta respecto a la inicialización del SSLContext en el programa?

- a. Utiliza exclusivamente certificados digitales de la autoridad de certificación.
- b. Requiere un trustStore para su inicialización.
- c. Utiliza un KeyManagerFactory para inicializar con el almacén de claves.
- d. No requiere de un almacén de claves para su operación.

36) ¿Qué puerto está configurado para escuchar las conexiones seguras en el servidor?

- a. 8080
- b. 443
- c. 8443
- d. 80

37) ¿Qué versión de TLS se especifica para el SSLContext?

- a. SSLv3
- b. TLSv1
- c. TLSv1.2
- d. No se especifica, se usa la predeterminada del sistema

38) Considerando el almacenamiento y manejo de contraseñas en el programa SecureServer, ¿cuál de las siguientes técnicas podría mejorar la seguridad del almacenamiento del keystore y la gestión de su contraseña?

- a. Almacenar la contraseña del keystore en un archivo de propiedades dentro del proyecto.
- b. Utilizar una función hash segura para almacenar la contraseña del keystore.
- c. Emplear una herramienta de gestión de secretos para almacenar y recuperar la contraseña del keystore en tiempo de ejecución.
- d. Mantener la contraseña del keystore codificada en base64 dentro del código fuente.

39) Dado que el programa SecureServer inicializa un SSLContext con TLS para comunicaciones seguras, ¿cuál de las siguientes opciones representa una best practice para fortalecer la seguridad de la conexión SSL/TLS?

- a. Limitar la negociación a versiones específicas de TLS, preferiblemente TLS 1.2 o superior.
- b. Permitir que el cliente use cualquier versión de SSL/TLS disponible, incluyendo SSLv3, para maximizar la compatibilidad.
- c. Usar siempre la versión predeterminada de TLS proporcionada por la JVM, sin especificar ninguna preferencia.
- d. Deshabilitar todas las suites de cifrado que no utilicen cifrado de clave pública.



Unión Europea

Fondo Social Europeo

"El FSE invierte en tu futuro"



IES Rey Fernando VI



Comunidad de Madrid

40) En el contexto de aceptar conexiones seguras, ¿cuál de las siguientes modificaciones al manejo de SSLSocket en el programa SecureServer alinearía mejor con los principios de diseño de software seguro?

- a. Implementar un timeout de conexión para cerrar automáticamente SSLSockets inactivos.
- b. Crear un nuevo hilo por cada SSLSocket aceptado para mejorar el rendimiento.
- c. Utilizar un bloque finally para asegurar que cada SSLSocket se cierre adecuadamente después de su uso.
- d. Capturar y registrar excepciones específicas de SSL para cada operación realizada sobre el SSLSocket.