CESUR
Tu Centro Oficial de FP

EXAMEN PARCIAL – UNIDAD 7	
TRIMESTRE: SEGUNDO	**Fecha:**

CICLO: Desarrollo de Aplicaciones Web.	*CURSO: 1º*	*CALIFICACIÓN:*
MÓDULO: Programación	*Turno: Mañana*	

Nombre:	***Apellidos:***

Instrucciones: *Esta prueba tiene como finalidad evaluar los aprendizajes de Programación. Lee atentamente y responde escribiendo el código más adecuado.*

Si las instrucciones no se siguen como se especifican el examen no será evaluado

PARTE PRÁCTICA. TIPO C.

- El examen práctico tiene una puntuación máxima de 10 puntos.
- Para superar la parte práctica se requiere alcanzar un mínimo de 5 puntos.

1. **(5 puntos)** Desarrollar un sistema de seguridad (autenticación) que permita a un usuario intentar acceder al sistema ingresando una contraseña. El sistema debe permitir hasta 3 intentos fallidos consecutivos antes de bloquear la cuenta. En cada intento fallido de debe aplicar un bloqueo que se incremente de forma exponencial en cada bloqueo consecutivo (por ejemplo, 30 segundos en el primer bloqueo, 60 segundos en el siguiente, 120 segundos, etc.).
   Registrar en un array (de tamaño fijo) la fecha/hora y el resultado ("Éxito" o "Fallo") de cada intento.
   Lanzar excepciones personalizadas en los siguientes casos:
   - **CredencialesIncorrectasException**: cuando la contraseña ingresada es incorrecta y aún no se han alcanzado los 3 intentos consecutivos.
   - **DemasiadosIntentosException**: cuando se acumulan 3 intentos fallidos consecutivos, activando el bloqueo.
   - **CuentaBloqueadaException**: cuando se intenta acceder durante el período de bloqueo.

   Clases:
   - **Intento**: registra los datos de cada intento de acceso.
     - Atributos:
       - **timestamp** (tipo long): Guarda la fecha y hora del intento (por ejemplo, utilizando System.currentTimeMillis()).
       - **resultado** (tipo String): Almacena el resultado del intento ("Éxito" o "Fallo").
     - Métodos:
       - **public String toString()**: Devuelve una cadena que muestre de forma legible la fecha/hora y el resultado del intento.
   - SistemaSeguridad: gestiona el proceso de autenticación, controlar los intentos fallidos y el bloqueo, y registra el historial de intentos.
     - Atributos:
       - **claveCorrecta** (tipo String): Almacena la contraseña válida (por ejemplo, "seguro123").
       - **consecutiveFailures** (tipo int): Contador de intentos fallidos consecutivos.
       - **lockExpirationTime** (tipo long): Indica hasta qué momento (en milisegundos) la cuenta estará bloqueada.
       - **baseLockDuration** (tipo long): Duración base del bloqueo en milisegundos (por ejemplo, 30000 ms para 30 segundos).

- **lockMultiplier** (tipo int): Multiplicador para la duración del bloqueo, que se incrementa (por ejemplo, se duplica en cada bloqueo consecutivo).
- **historial** (tipo Intento[]): Array de tamaño fijo (por ejemplo, de 10 registros) para almacenar los intentos de acceso.
- **numIntentos** (tipo int): Número actual de intentos registrados en el historial.

- Métodos:
  - **public void acceder(String clave)**: Verifica si la cuenta está bloqueada comparando el tiempo actual con lockExpirationTime. Si está bloqueada, lanzar CuentaBloqueadaException indicando el tiempo restante de bloqueo. Si la **contraseña es correcta**, registrar el intento como "Éxito", reiniciar el contador de intentos fallidos y el multiplicador de bloqueo y conceder el acceso.
    Si la **contraseña es incorrecta**, incrementar consecutiveFailures, registrar el intento como "Fallo" y, **si se alcanza 3 intentos fallidos** consecutivos, establecer lockExpirationTime con la duración calculada (baseLockDuration * lockMultiplier), incrementar el lockMultiplier, reiniciar el contador de fallos y lanzar DemasiadosIntentosException, **si no se alcanzan 3 fallos**, lanzar CredencialesIncorrectasException indicando el número de intento fallido.
  - **private void registrarIntento(String resultado)**: agregar un nuevo objeto Intento al array de historial. Si el array está lleno, se debe desplazar o descartar el intento más antiguo para dar cabida al nuevo.
  - **public void mostrarHistorial()**: imprimir en consola todos los registros del historial de intentos.

Prueba tu programa con la clase Test:

```
public class Test {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        SistemaSeguridad seguridad = new SistemaSeguridad();

        // Bucle de intentos hasta que se logre un acceso exitoso.
        while (true) {
            System.out.print("Ingrese la contraseña: ");
            String clave = scanner.nextLine();

            try {
                seguridad.acceder(clave);
                // Salir del bucle en caso de éxito.
                break;
            } catch (CuentaBloqueadaException e) {
                System.out.println("⚠️ " + e.getMessage());
            } catch (CredencialesIncorrectasException | DemasiadosIntentosException e) {
                System.out.println("❌ " + e.getMessage());
            }
        }

        // Mostrar el historial de intentos tras el acceso exitoso.
        seguridad.mostrarHistorial();
        scanner.close();
    }
}
```

Ejemplo de ejecución:

```
Ingrese la contraseña: 5416135
❌ Contraseña incorrecta. Intento 1 de 3.
Ingrese la contraseña: seguro888
❌ Contraseña incorrecta. Intento 2 de 3.
Ingrese la contraseña: seguro123
¡Acceso concedido!

Historial de intentos:
Intento en 1739433079566 ms - Resultado: Fallo
Intento en 1739433095365 ms - Resultado: Fallo
Intento en 1739433097724 ms - Resultado: Éxito
```

2. **(5 puntos)** Desarrollar un sistema de control de acceso para un parking que permita gestionar la entrada y salida de vehículos. Cada vehículo se identificará mediante su matrícula y se registrará la hora de entrada. El sistema debe validar condiciones para evitar errores, usando arrays para almacenar la información.

Clases

- Vehículo
  - Atributos:
    - **matricula** (String): La matrícula del vehículo.
    - **horaEntrada** (LocalDateTime): La hora en la que el vehículo ingresó al parking.
  - Métodos:
    - **public Vehiculo(String matricula):** Crea un vehículo asignándole la matrícula recibida y registrando la hora de entrada (usando, por ejemplo, LocalDateTime.now()).
    - **toString()**: Devuelve una cadena con la información del vehículo (matrícula y hora de entrada) de forma legible.
- Parking
  - Atributos:
    - **vehiculos** (array de Vehiculo): Array de tamaño fijo que almacena los vehículos actualmente estacionados (por ejemplo, capacidad máxima 20).
    - **numVehiculos** (int): Contador de vehículos actualmente en el parking.
    - **CAPACIDAD_MAXIMA** (constante int): Capacidad máxima del parking.
  - Métodos:
    - **ingresarVehiculo(String matricula)**: Verifica si el parking está lleno. Si es así, lanzar ParkingLlenoException. Verifica si ya existe un vehículo con la misma matrícula. Si existe, lanzar VehiculoDuplicadoException. En caso contrario, crear un nuevo objeto Vehiculo (registrando la hora de entrada) y agregarlo al array.
    - **salirVehiculo(String matricula)**:Busca en el array un vehículo con la matrícula indicada. Si no se encuentra, lanzar VehiculoNoEncontradoException. Si se encuentra, eliminar el

vehículo del array (se puede desplazar el resto de los elementos para no dejar "huecos") y actualizar el contador.

- **mostrarVehiculos()**: Recorre el array y mostrar la matrícula y la hora de entrada de cada vehículo actualmente estacionado.

- **Excepciones Personalizadas**:
  - **ParkingLlenoException**:
    - Constructor: public ParkingLlenoException(String mensaje)
  - **VehiculoDuplicadoException**:
    - Constructor: public VehiculoDuplicadoException(String mensaje)
  - **VehiculoNoEncontradoException**:
    - Constructor: public VehiculoNoEncontradoException(String mensaje)

Para almacenar la hora de entrada utilizaremos de la clase LocalDateTime el método LocalDateTime.now()

El método toString() de la clase Transaccion, puede ser este:

```
@Override
public String toString() {
    // Formateamos la hora de entrada para una visualización amigable.
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
    return "Matrícula: " + matricula + " | Hora de entrada: " + horaEntrada.format(formatter);
}
```

Prueba tu programa con la clase Test:

```
public class Test {
    public static void main(String[] args) {
        // Crear un parking con capacidad máxima de 5 vehículos para pruebas.
        Parking parking = new Parking( capacidad: 5);
        Scanner scanner = new Scanner(System.in);
        boolean continuar = true;

        // Menú interactivo simple.
        while (continuar) {
            System.out.println("\n--- Menú del Parking ---");
            System.out.println("1. Ingresar vehículo");
            System.out.println("2. Salir vehículo");
            System.out.println("3. Mostrar estado del parking");
            System.out.println("4. Salir del sistema");
            System.out.print("Seleccione una opción: ");
            int opcion = scanner.nextInt();
            scanner.nextLine();  // Limpiar el buffer.

            switch (opcion) {
                case 1:
                    System.out.print("Ingrese la matrícula del vehículo: ");
                    String matriculaIngreso = scanner.nextLine();
                    try {
                        parking.ingresarVehiculo(matriculaIngreso);
                    } catch (ParkingLlenoException | VehiculoDuplicadoException e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;
                case 2:
                    System.out.print("Ingrese la matrícula del vehículo a retirar: ");
                    String matriculaSalida = scanner.nextLine();
                    try {
                        parking.salirVehiculo(matriculaSalida);
                    } catch (VehiculoNoEncontradoException e) {
                        System.out.println("Error: " + e.getMessage());
                    }
                    break;
                case 3:
                    parking.mostrarVehiculos();
                    break;
                case 4:
                    continuar = false;
                    default:
                        System.out.println("Opción no válida. Intente nuevamente.");
                }
            }
            scanner.close();
        }
    }
```

Ejemplo de ejecución:

```
--- Menú del Parking ---
1. Ingresar vehículo
2. Salir vehículo
3. Mostrar estado del parking
4. Salir del sistema
Seleccione una opción: 1
Ingrese la matrícula del vehículo: 1234AAA
Vehículo con matrícula 1234AAA ingresado correctamente.

--- Menú del Parking ---
1. Ingresar vehículo
2. Salir vehículo
3. Mostrar estado del parking
4. Salir del sistema
Seleccione una opción: 1
Ingrese la matrícula del vehículo: 5678ZZZ
Vehículo con matrícula 5678ZZZ ingresado correctamente.

--- Menú del Parking ---
1. Ingresar vehículo
2. Salir vehículo
3. Mostrar estado del parking
4. Salir del sistema
Seleccione una opción: 3

Estado actual del parking:
Matrícula: 1234AAA | Hora de entrada: 2025-02-13 11:35:03
Matrícula: 5678ZZZ | Hora de entrada: 2025-02-13 11:35:13
```