CESUR
Tu Centro Oficial de FP

EXAMEN PARCIAL – UNIDAD 7		
TRIMESTRE: SEGUNDO		**Fecha:**
CICLO: Desarrollo de Aplicaciones Web *MÓDULO: Programación*	*CURSO: 1º* *Turno: Mañana*	*CALIFICACIÓN:*
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 B.

- 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)** Implementar un sistema de gestión de contraseñas para un usuario, que permita actualizar su contraseña cumpliendo ciertos requisitos de seguridad y evitando la reutilización de las últimas tres contraseñas utilizadas.

   Validación de la nueva Contraseña:
   - La contraseña debe tener al menos 8 caracteres.
   - Debe incluir al menos una letra mayúscula.
   - Debe incluir al menos una letra minúscula.
   - Debe incluir al menos un dígito numérico.
   - Debe incluir al menos un carácter especial (un carácter que no sea una letra ni un dígito).

   Historial de Contraseñas:
   - Se mantendrá un historial de las últimas 3 contraseñas utilizadas (incluyendo la actual).
   - Si la nueva contraseña coincide con la actual o con alguna de las últimas 3 contraseñas usadas, se lanzará una excepción.

   Excepciones Personalizadas:
   - **PasswordInvalidaException**: Se lanza cuando la nueva contraseña no cumple con los requisitos de seguridad.
   - **PasswordRepetidaException**: Se lanza cuando la nueva contraseña es igual a la actual o a alguna de las últimas 3 contraseñas en el historial.

   Actualización de la Contraseña:
   - Si la contraseña es válida y no se ha utilizado recientemente, se actualiza la contraseña actual.
   - Si ya existe una contraseña previa, ésta se mueve al historial.
   - El historial se gestiona de forma FIFO (First In, First Out): cuando se alcanza la capacidad máxima (3 contraseñas), la más antigua se descarta para dejar espacio a la nueva.

Clases que tenéis que generar:

PasswordInvalidaException: Representa el error que ocurre cuando la contraseña no cumple con los requisitos de seguridad. Debe tener un constructor que reciba un mensaje de error.

PasswordRepetidaException: Representa el error que ocurre cuando la nueva contraseña coincide con la actual o con alguna del historial. Debe tener un constructor que reciba un mensaje de error.

Usuario: Gestionará la contraseña del usuario y el historial de contraseñas.

- Atributos:
  - **password** (String): Contraseña actual del usuario.
  - **historialPasswords** (array de String): Array de tamaño fijo para almacenar las últimas 3 contraseñas.
  - **numHistorial** (int): Contador que indica cuántas contraseñas se han almacenado en el historial.
- Métodos:
  - **setPassword(String nuevaPassword)**: Valida la nueva contraseña y, de ser correcta, la asigna como la contraseña actual. Si había una contraseña previa, la mueve al historial. Debe lanzar PasswordInvalidaException o PasswordRepetidaException según corresponda.
  - **mostrarHistorial()**: Muestra por consola el contenido del historial de contraseñas almacenado en el array.

Para testear si un carácter está en mayúsculas, minúsculas o es un número, tenemos en la clase Character los métodos isUpperCase(ch), isLowerCase(ch), isDigit(ch). Si no es nada de esto, consideramos que es un carácter especial.

Prueba tu programa con la clase Test:

```
public class Test {
    public static void main(String[] args) {
        Usuario usuario = new Usuario();
        try {
            // Establecer la primera contraseña (válida).
            usuario.setPassword("Abcdef1!");
            // Actualizar a una nueva contraseña válida.
            usuario.setPassword("Xyz123@#");
            // Actualizar a otra contraseña válida.
            usuario.setPassword("MnbvcZ9$");
            // Intentar establecer una contraseña que ya se usó anteriormente (debe lanzar excepción).
            usuario.setPassword("Abcdef1!");
        } catch (PasswordInvalidaException | PasswordRepetidaException e) {
            System.out.println("Error: " + e.getMessage());
        }
        // Mostrar el historial de contraseñas.
        usuario.mostrarHistorial();
    }
}
```

Ejemplo de ejecución:

```
Contraseña actualizada correctamente.
Contraseña actualizada correctamente.
Contraseña actualizada correctamente.
Error: La contraseña no puede ser igual a alguna de las últimas 3 utilizadas.
Historial de contraseñas:
  - Abcdef1!
  - Xyz123@#
```

2. **(5 puntos)** Implementar un sistema que permita gestionar la información de personas. Se debe validar que la edad y el DNI sean correctos y actualizar los datos de una persona mediante métodos de validación. Además, se implementará una clase para gestionar un registro de personas, evitando duplicados en el DNI. Tienes que implementar:
    - Clase **Persona**
        - Atributos
            - **nombre** (String): Nombre completo de la persona.
            - **edad** (int): Edad de la persona.
            - **dni** (String): Documento Nacional de Identidad.
        - Métodos
            - **setEdad(int edad)**:Valida que la edad esté entre 0 y 150. Si no se cumple, lanza la excepción EdadInvalidaException.
            - **setDni(String dni)**: Valida que el DNI tenga exactamente 8 dígitos seguidos de una letra mayúscula (por ejemplo, "12345678A"). Si el formato es incorrecto, lanza la excepción DniInvalidoException.
            - **esMayorDeEdad()**: Devuelve true si la edad es mayor o igual a 18, o false en caso contrario.
            - **actualizarDatos(String nombre, int edad, String dni)**: Actualiza el nombre, la edad y el DNI de la persona utilizando los métodos de validación anteriores. Lanza la excepción correspondiente si algún dato no es válido.
    - Clase **RegistroPersonas**: Gestiona un array de objetos de tipo Persona.
        - Atributos
            - **personas** (array de Persona): Array de tamaño fijo para almacenar las personas (por ejemplo, 10).
            - **numPersonas** (int): Número de personas actualmente registradas.
        - Métodos
            - **agregarPersona(Persona p)**: Agrega una nueva persona al registro. Si el DNI de la persona ya existe en el registro, lanza DniDuplicadoException.
            - **buscarPorNombre(String nombre)**: Busca y devuelve (o muestra) las personas cuyo nombre contenga el texto indicado.
            - **listarPersonas()**: Muestra la lista de todas las personas registradas.

- Excepciones
    - **EdadInvalidaException**: Se lanza cuando la edad proporcionada es menor que 0 o mayor que 150.
    - **DniInvalidoException**: Se lanza cuando el DNI no cumple el formato (8 dígitos seguidos de una letra mayúscula).
    - **DniDuplicadoException**: Se lanza cuando se intenta agregar a una persona cuyo DNI ya existe en el registro.

Prueba tu programa con la clase Test:

```
public class Test {
    public static void main(String[] args) {
        try {
            // Crear instancias de Persona con datos válidos.
            Persona p1 = new Persona( nombre: "Ana García",  edad: 25,  dni: "12345678A");
            Persona p2 = new Persona( nombre: "Luis Martínez",  edad: 17,  dni: "87654321B");
            Persona p3 = new Persona( nombre: "Carlos López",  edad: 45,  dni: "11223344C");
            // Mostrar datos y comprobación de mayoría de edad.
            System.out.println(p1);
            System.out.println(p2);
            System.out.println(p3);
            System.out.println();
            // Actualizar datos de p2 (por ejemplo, cambiar edad y DNI).
            p2.actualizarDatos( nombre: "Luis Martínez",  edad: 18,  dni: "87654321B");  // Cambiamos la edad a 18
            System.out.println("Después de actualizar datos de p2:");
            System.out.println(p2);
            System.out.println();
            // Crear un registro de personas.
            RegistroPersonas registro = new RegistroPersonas();
            // Intentar agregar una persona con DNI duplicado (esto lanzará excepción).
            try {...} catch (DniDuplicadoException e) {
                System.out.println("Error al agregar persona: " + e.getMessage());
            }
            // Buscar personas por nombre.
            registro.buscarPorNombre( nombreFragment: "Martí");
            // Listar todas las personas registradas.
            System.out.println();
            registro.listarPersonas();
        } catch (EdadInvalidaException | DniInvalidoException e) {
            System.out.println("Error en la creación de la persona: " + e.getMessage());
        }
    }
}
```

Ejemplo de ejecución:

```
Nombre: Ana García | Edad: 25 | DNI: 12345678A | Mayor de edad
Nombre: Luis Martínez | Edad: 17 | DNI: 87654321B | Menor de edad
Nombre: Carlos López | Edad: 45 | DNI: 11223344C | Mayor de edad

Después de actualizar datos de p2:
Nombre: Luis Martínez | Edad: 18 | DNI: 87654321B | Mayor de edad

Error al agregar persona: El DNI 12345678A ya está registrado.
Resultados de la búsqueda por nombre 'Martí':
Nombre: Luis Martínez | Edad: 18 | DNI: 87654321B | Mayor de edad

Listado de todas las personas registradas:
Nombre: Ana García | Edad: 25 | DNI: 12345678A | Mayor de edad
Nombre: Luis Martínez | Edad: 18 | DNI: 87654321B | Mayor de edad
Nombre: Carlos López | Edad: 45 | DNI: 11223344C | Mayor de edad
```