Ejercicio práctico 3.9 - fechas y expresiones regulares

El proyecto esta dividido en **dos grandes bloques**:

- `com.docencia.logica` → logica usando API `java.time`
- `com.docencia.expresiones` → combinacion de **regex + fechas**

Cada ejercicio esta en su propio paquete:

```
com.docencia.fechas.ejercicioN
com.docencia.logica.ejercicioN
```

y contiene:

- `EjercicioN.java` → implementacion
- `EjercicioNTest.java` → 10 pruebas unitarias

Rúbrica

La evaluacion suele hacerse por clases (no por numero total de tests).

- 0 tests incorrectos 1 punto.
- 1-2 test incorrectos se da el 0.75 puntos.

---

com.docencia.expresiones.Ejercicio1

Implementar y probar un método que valide fechas en formato **dd/mm/aaaa** usando una **expresión regular**.

La validación debe comprobar:

- **Día**: de `01` a `31`
- **Mes**: de `01` a `12`
- **Año**: exactamente **4 dígitos** (`0000` a `9999`)

> Importante: esta validación es **solo por rango**, no valida calendario real. Por ejemplo, `31/04/2026` pasará (porque 31 y 04 están dentro de rango), aunque abril no tenga 31 días.

com.docencia.expresiones.Ejercicio2

Implementar y probar un método que valide URLs que empiecen por **http://** o **https://**, tengan un **dominio válido** y permitan una **ruta opcional**.

Se valida:

- **Protocolo**: `http://` o `https://`
- **Dominio**:

    - Uno o más labels separados por puntos (ej: `midominio.com`, `api.midominio.com`)
    - Cada label permite letras, números y guiones (`A-Za-z0-9-`)
    - Debe terminar en un **TLD** solo letras de longitud **2 a 63** (ej: `.com`, `.es`, `.travel`)
- **Ruta opcional**:
    - Puede ir después del dominio como `/algo/...`
    - No se permiten espacios en la ruta

> Nota: esta regex no valida punycode/IDN (dominios con caracteres no ASCII), ni comprueba IPs, puertos, querystring detallado, etc. Es una validación "práctica" para examen.

com.docencia.expresiones.Ejercicio3

Implementar y probar un método que valide si una contraseña es **fuerte** según estas reglas:

- Longitud mínima: **10 caracteres**
- Debe contener al menos:
    - **1 mayúscula** (`A-Z`)
    - **1 minúscula** (`a-z`)
    - **1 número** (`0-9`)
    - **1 símbolo** (cualquier carácter que NO sea letra ni número)
- **No se permiten espacios** (ni al inicio, ni en medio, ni al final)

com.docencia.expresiones.Ejercicio4

Implementar y probar un método que valide strings con este formato:

- **Solo fecha ISO**: `aaaa-mm-dd`
- **Fecha + hora opcional**: `aaaa-mm-dd HH:MM`

Reglas:

- Año: exactamente **4 dígitos**
- Mes: `01` a `12`
- Día: `01` a `31` (solo rango, NO calendario real)
- Si aparece hora:
    - Hora: `00` a `23`
    - Minutos: `00` a `59`
- Separador entre fecha y hora: **un espacio**
- La hora es **opcional** (puede no estar)

> Importante: esta regex valida rango, no "fecha real" (por ejemplo, `2026-04-31` pasaría).

com.docencia.expresiones.Ejercicio5

Implementar y probar un método que valide strings con este formato:

`dd/mm/aaaa h:MM AM|PM`

Ejemplos de formato:

- `13/02/2026 9:30 AM`

- `13/02/2026 12:00 PM`
- `01/12/1999 09:05 PM` (también se permite `09` por el `0?`)

Reglas que se validan:

- Día: `01` a `31`
- Mes: `01` a `12`
- Año: exactamente 4 dígitos
- Un espacio entre fecha y hora
- Hora (12h): `1` a `12` (se permite `9` o `09`; y `12`)
- Minutos: `00` a `59`
- Un espacio antes de `AM` o `PM`
- `AM` o `PM` debe estar en mayúsculas (exacto)

> Importante: como es regex, valida solo rango; no valida calendario real (por ejemplo `31/04/2026` pasaría).

com.docencia.logica.Ejercicio1

Implementar y probar un método que:

1. Reciba una hora en formato **HH:mm** (24h).
2. Valide que la hora existe (por ejemplo, `24:00` es inválida).
3. Compruebe si la hora está dentro del intervalo **[inicio, fin]** (incluido).

> Asunción del ejercicio: `inicio <= fin` (intervalo normal del mismo día).

com.docencia.logica.Ejercicio2

Implementar y probar un método que calcule la **edad exacta** (en años) a partir de una fecha de nacimiento.

com.docencia.logica.Ejercicio3

Implementar y probar un método que devuelva el **primer día laborable** a partir de una fecha dada (incluyéndola), saltando fines de semana y festivos.

com.docencia.logica.Ejercicio4

Implementar y probar un método que indique si una fecha (`LocalDate`) es **laborable**.

com.docencia.logica.Ejercicio5

Sumar `n` días laborables (lunes-viernes) a una fecha inicial.

Reglas

- `fechaInicio` formato `dd/MM/uuuu` con parseo estricto.
- Si `fechaInicio` es `null` → lanzar `IllegalArgumentException`.
- Si `n < 0` → lanzar `IllegalArgumentException`.
- Si `n == 0` → devolver la misma fecha (aunque sea fin de semana).

- Para `n > 0`: se avanza día a día; solo cuentan días laborables (lunes-viernes).
- No se consideran festivos, únicamente fin de semana.

Ejemplos

- `fechaInicio="13/02/2026" (viernes), n=1` → `2026-02-16` (lunes)
- `fechaInicio="13/02/2026" (viernes), n=2` → `2026-02-17` (martes)
- `fechaInicio="14/02/2026" (sábado), n=1` → `2026-02-16` (lunes)
- `fechaInicio="16/02/2026" (lunes), n=5` → `2026-02-23` (lunes)
- `fechaInicio="31/04/2026", n=1` → error (fecha inválida)