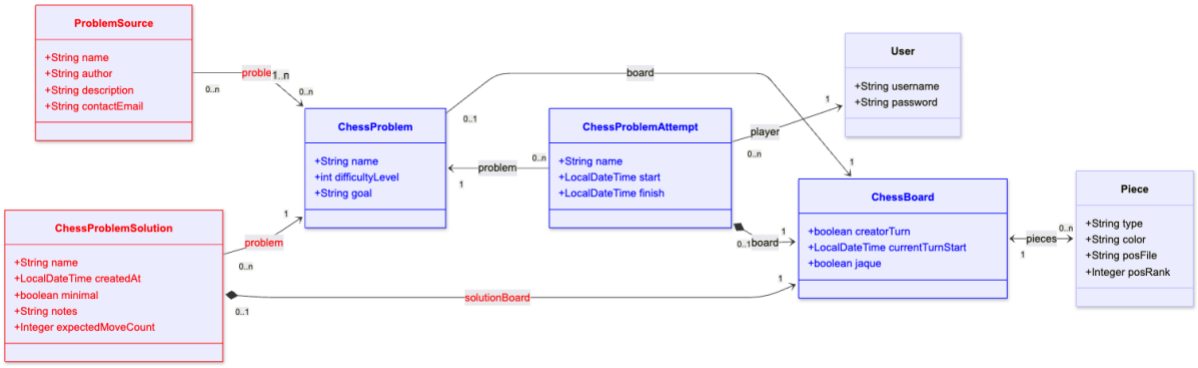
Control práctico de DP1 2025-2026 (Control-check 1)

Enunciado

En este ejercicio añadiremos la funcionalidad de **gestión de problemas de ajedrez** dentro de una aplicación web de ajedrez.

Concretamente, se proporciona una clase *ChessProblem* que representa un problema de ajedrez que los usuarios de la plataforma deben resolver. Un problema de ajedrez es un reto o enigma basado en el ajedrez, creado con una posición específica en el tablero (*ChessBoard*), que tiene un objetivo definido (por ejemplo, "dar jaque mate en dos jugadas") y que se debe resolver siguiendo las reglas del ajedrez. Los problemas de ajedrez parten de una posición inicial y tienen una o varias soluciones posibles. Cada solución está representada por la clase *ChessProblemSolution* y consiste en un tablero que representa la solución final (*solutionBoard)* y un número de movimientos esperados (*expectedMoveCount*). Los problemas pueden tener una fuente de dónde se han tomado, por ejemplo un libro. Esta fuente está represantada por la clase *ProblemSource*. Por último, la clase ChessProblemAttempt representa un intento de resolver el problema por parte de un usuario (clase *User*).

El diagrama UML que describe las clases y relaciones con las que vamos a trabajar es el siguiente:



Las clases para las que realizaremos el mapeo objeto-relacional como entidades JPA se han señalado en rojo. Las clases en azul son clases que se proporcionan ya mapeadas, pero con las que se trabajará durante el control de laboratorio.

Realizaremos una serie de ejercicios basados en funcionalidades que implementaremos en el sistema, y validaremos mediante pruebas unitarias. Si desea ver el resultado que arrojarían las pruebas en backend, puede ejecutarlas (bien mediante su entorno de desarrollo favorito, bien mediante el comando "*mvnw test*" en la carpeta raíz del proyecto). Cada ejercicio correctamente resuelto valdrá dos puntos, el número de casos de prueba de cada ejercicio puede variar entre uno y otro y la nota se calculará en base al porcentaje de casos de prueba que pasan. Por ejemplo, si pasan la mitad (50%) de los casos de prueba de un ejercicio, usted obtendrá un punto (2*0,5=1), y si pasan un 10% obtendrá 0,2 puntos.

La entrega del control check se realizará en dos pasos:

1. **Entregando la actividad en EV asociada al control check proporcionando como texto la dirección url de su repositorio personal.** Puede hacer esto al comienzo del control.

2. **Mediante un único comando "*git push*" a su repositorio individual donde se encuentra este documento una vez ha finalizado de completar los ejercicios.**

Recuerde que debe hacer push antes de cerrar sesión en la computadora y abandonar el aula, de lo contrario, su intento se evaluará como no presentado. Su primera tarea en este control será clonar el repositorio (recuerde que si va a usar los equipos del aula para realizar el control necesitará usar un token de autenticación de GitHub como clave, tiene un documento de ayuda a la configuración en el propio repositorio del control). A continuación, deberá importar el proyecto en su entorno de desarrollo favorito y comenzar los ejercicios abajo listados. Al importar el proyecto, el mismo puede presentar errores de compilación. No se preocupe, si existen, dichos errores irán despareciendo conforme usted vaya implementando los distintos ejercicios del control.

Nota importante 1: No modifique los nombres de las clases ni la signatura (nombre, tipo de respuesta y parámetros) de los métodos proporcionados como material de base para el control. Las pruebas que se usan para la evaluación dependen de que las clases y los métodos tengan la estructura y nombres proporcionados. Si los modifica probablemente no pueda hacer que pasen las pruebas.

Nota importante 2: No modifique las pruebas unitarias proporcionadas como parte del proyecto bajo ningún concepto. Aunque modifique las pruebas en su copia local del proyecto, éstas serán restituidas mediante un comando git previamente a la ejecución de las pruebas para la emisión de la nota final, por lo que sus modificaciones en las pruebas no serán tenidas en cuenta en ningún momento.

Nota importante 3: Mientras haya ejercicios no resueltos habrá tests que no funcionen y, por tanto, el comando "*mvnw install*" finalizará con error. Esto es normal debido a la forma en la que está planteado el control y no hay que preocuparse por ello. Si se quiere probar la aplicación se puede ejecutar de la forma habitual pese a que "mvnw install" finalice con error.

Nota importante 4: La descarga del material de la prueba usando git, y la entrega de su solución con git a través del repositorio GitHub creado a tal efecto forman parte de las competencias evaluadas durante el examen, por lo que no se aceptarán entregas que no hagan uso de este medio, y no se podrá solicitar ayuda a los profesores para realizar estas tareas.

Nota importante 5: No se aceptarán como soluciones válidas proyectos cuyo código fuente no compile correctamente o que provoquen fallos al arrancar la aplicación en la inicialización del contexto de Spring. Las soluciones cuyo código fuente no compile o incapaces de arrancar el contexto de Spring serán evaluadas con una nota de 0.

Nota importante 6: Excepto el ejercicio 5, que depende del 4, lo ejercicios del examen son independientes y pueden ser resueltos en cualquier orden.

Test 1 – Modificar el servicio de *ChessProblemAttempts* para que no permita guardar partidas inválidas (regla de negocio) (2 puntos)

Modificar el método save del servicio de gestión de partidas (*ChessProblemAttemptService*) de manera que se lance la excepción (*UnfeasibleAttemptException*) en caso de que se intente guardar una partida que incumpla alguna de las siguientes reglas de negocio:

- No debe haber ningún otro intento de resolución del mismo problema activo en ese momento (es decir, cuyo instante de fin sea nulo) por parte del mismo jugador a no ser que se trate del intento de resolución que estamos guardando en ese momento. En el repositorio *ChessProblemAttemptRepository* puedes encontrar un método que te ayuda a evaluar esta regla de negocio.
- El instante de inicio debe ser anterior al instante de fin en caso de que este no sea nulo.

El método debe ser transaccional y hacer *rollback* de cualquier cambio si se lanza dicha excepción.

Test 2 – Creación de servicio y repositorio de *ChessProblem* y del controlador de *ChessProblemAttempt*

Parte 2A: Servicio y repositorio de *ChessProblem* (1 punto):

Modificar la clase *ChessProblemService* y la interfaz *ChessProblemRepository*, para que sean un servicio Spring de lógica de negocio y un repositorio de Spring Data. Además, se debe proporcionar una implementación a los métodos del servicio usando el repositorio, que debe ser inyectado mediante Spring:

Para el caso del servicio de *ChessProblem* debe permitirse:

1. Obtener todos los problemas de un nivel de dificultad dado o superior y que le queden menos piezas en el tablero de las dadas (método *getProblemsByDifficultyAndNumPieces(Integer difficulty, Integer numPieces)*).
2. Obtener todos los problemas existentes (método *getAll*).
3. Obtener un problema concreto por id (método *getById*).
4. Eliminar un problema de la base de datos (método *delete*).

Todos estos métodos de la clase *ChessProblemService* **deben ser transaccionales**, pero las anotaciones asociadas deben realizarse a nivel de método, no a nivel de clase**.**

El repositorio deberá implementar las consultas correspondientes que considere necesarias.

Parte 2B: Controlador para API de *ChessProblemAttempt* (1 punto):

Configurar la clase *ChessProblemAttemptController* como un controlador y crear un método que permita devolver todos los intentos existentes. El método debe responder a peticiones tipo GET en la URL:

http://localhost:8080/api/v1/attempts

Igualmente debe crear un método para devolver un intento concreto, este método debe responder a peticiones en la url http://localhost:8080/api/v1/attempts/X donde X es la Id del intento a obtener, y

devolverá los datos del intento correspondiente. En caso de que no exista un intento con el id especificado el sistema debe devolver el código de estado 404 (NOT_FOUND).

Estos endpoint de la API asociados a la gestión de intentos deberían estar accesibles únicamente para usuarios de tipo Player (que tengan la authority "PLAYER"), devolviendo un código 403 (FORBIDDEN) en caso contrario.

Test 3 – Creación de un componente React de listado de intentos (2 puntos)

Modificar el componente React proporcionado en el fichero *frontend/src/attempts/index.js* para que muestre un listado de las partidas disponibles en el sistema. Para ello debe hacer uso de la API lanzando una petición tipo GET contra la URL /api/v1/attempts.

Si el conjunto de partidas no está vacío, este componente debe mostrar los intentos en una tabla (se recomienda usar el componente *Table* de *reactstrap*). Los títulos de las cabeceras de las columnas de la tabla deben ser "Name", "Start", "Finish", "Player", "Problem" y "Difficulty" (en ese orden). Las columnas debe incluir los valores del nombre del intento, sus fechas de inicio y fin (si la hubiera), el nombre del jugador, el nombre del problema y su nivel de dificultad.

Si el conjunto de intentos de resolución de problema devuelto por la API es un array vacío, el componente debe mostrar un título de nivel 1 (h1), con el texto "NO ATTEMPTS", en lugar de la tabla (no debe aparecer la tabla vacía).

Para poder lanzar esta prueba y comprobar su resultado puede colocarse en la carpeta de frontend y ejecutar el comando npm test y pulsar 'a' en el menú de comandos de jest. Nótese que previamente debe haber lanzado al menos una vez el comando npm install para que todas las librerías de node estén instaladas.

Test 4 – Creación de las entidades *ChessProblemSolution* y *ProblemSource* y sus repositorios asociados

Parte 4A: Entidades y repositorios (1 punto)

Modificar las clases *ChessProblemSolution* y *ProblemSource* para que sean entidades. Estas clases deben tener los siguientes atributos y restricciones:

Para la clase *ChessProblemSolution*:

- El atributo de tipo entero (*Integer*) llamado "*id*" actuará como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- El atributo de tipo cadena de caracteres (*String*) llamado "*name*" obligatorio (no puede ser nulo), que debe tener una longitud mínima de 3 caracteres y máxima de 50 y que no puede estar formada por caracteres vacíos (espacios, tabuladores, etc.).
- El atributo de tipo instante temporal (*LocalDateTime*) llamado "*createdAt*" obligatorio, que representa el instante en que se creó la solución.
- El atributo de tipo *boolean* llamado "*minimal*" obligatorio, que indica si esta solución es la mínima (la más corta) del problema.

- El atributo de tipo cadena de caracteres (*String*) llamado "*notes*" opcional, que debe tener una longitud máxima de 4000 caracteres.
- El atributo de tipo entero (*Integer*) llamado "*expectedMoveCount*" que representa el número esperado de movimientos para alcanzar el objetivo. Este *atributo es opcional* y tendrá un *valor mínimo de 1 y máximo de 15*.

Para la clase *ProblemSource*:

- El atributo de tipo entero (*Integer*) llamado "*id*" actuará como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- El atributo de tipo cadena de caracteres (*String*) llamado "*name*" obligatorio (no puede ser nulo), que debe tener una longitud mínima de 3 caracteres y máxima de 50 y que no puede estar formada por caracteres vacíos (espacios, tabuladores, etc.).
- El atributo de tipo cadena de caracteres (*String*) llamado "*author*" obligatorio, que debe tener una longitud mínima de 3 caracteres y máxima de 100 caracteres.
- El atributo de tipo cadena de caracteres (*String*) llamado "*description*" opcional, que debe tener una longitud máxima de 1000 caracteres.
- El atributo de tipo cadena de caracteres (*String*) llamado "*contactEmail*" opcional, que debe ser una dirección de correo electrónico válida.

Modificar las interfaces *ChessProblemSolutionRepository* y *ProblemSourceRepository* alojadas en el mismo paquete para que extiendan a *CrudRepository*. No olvide especificar sus parámetros de tipo.

Parte 4B: Relaciones/asociaciones entre entidades (1 punto)

Elimine las anotaciones @Transient de los métodos y atributos que las tengan en las entidades creadas anteriormente. Se pide crear las siguientes relaciones entre las entidades. Cree una relación unidireccional desde *ChessProblemSolution* hacia *ChessProblem* y otra desde *ChessProblemSolution* hacia *ChessBoard* que expresen las que aparecen en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades.

Además, se pide crear una relación unidireccional desde *ProblemSource* hacia *ChessProblem* que represente la que aparece en el diagrama UML teniendo en cuenta su cardinalidad y usando el atributo *problems* en la clase *ProblemSource*.

Importante: En todos los casos debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, algunos atributos podrían ser nulos puesto que la cardinalidad es 0..n pero otros no, porque su cardinalidad en el extremo navegable de la relación es 1..n.

Test 5 – Modificación del script de inicialización de la base de datos para incluir dos *ChessProblemSolution*, dos *ProblemSource* y sus relaciones

Modificar el script de inicialización de la base de datos, para que se creen las siguientes soluciones (*ChessProblemSolution*) y orígenes de problemas (*ProblemSource*):

ProblemSource 1:

- **id:** 801

- **name:** “Classic Puzzle Compendium”
- **author:** “H. Steinitz”
- **description:** “A curated set of elegant checkmate studies published in 1895.”
- **contactEmail:** “steinitz@classicpuzzles.org”

ProblemSource 2:

- **id:** 802
- **name:** “Modern Engine Analysis 2025”
- **author:** “DeepEval AI”
- **description:** “Computer-validated solutions using latest neural networks.”
- **contactEmail:** “analysis@deepeval.ai”

ChessProblemSolution 1:

- **id:** 901
- **name:** “Canonical Mate in 1 (line A)”
- **createdAt:** 10-01-2025 09:11:00
- **minimal:** true
- **notes:** “Immediate mate with Qh5# from the starting position.”
- **expectedMoveCount:** 1

ChessProblemSolution 2:

- **id:** 902
- **name:** “Canonical Mate in 2 (line B)”
- **createdAt:** 10-01-2025 09:21:00
- **minimal:** true
- **notes:** “Two-move forcing sequence ending in mate with Qd5#.”
- **expectedMoveCount:** 2

Además, debe modificar este script de inicialización de la base de datos para que:

- El *ProblemSource* cuyo id es 801 se asocie con los problemas con id 1 y 3
- El *ProblemSource* cuyo id es 802 se asocie con los problemas con id 2, 4 y 5
- El *ChessProblemSolution* cuyo id es 901 tenga como problema asociado aquel cuyo id es 1.
- El *ChessProblemSolution* cuyo id es 901 tenga como solución el tablero con id 910.
- El *ChessProblemSolution* cuyo id es 902 tenga como problema asociado aquel cuyo id es 3.
- El *ChessProblemSolution* cuyo id es 902 tenga como solución el tablero con id 911.

Tenga en cuenta que el orden en que aparecen los INSERT en el script de inicialización de la base de datos es relevante al definir las asociaciones.