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

Enunciado

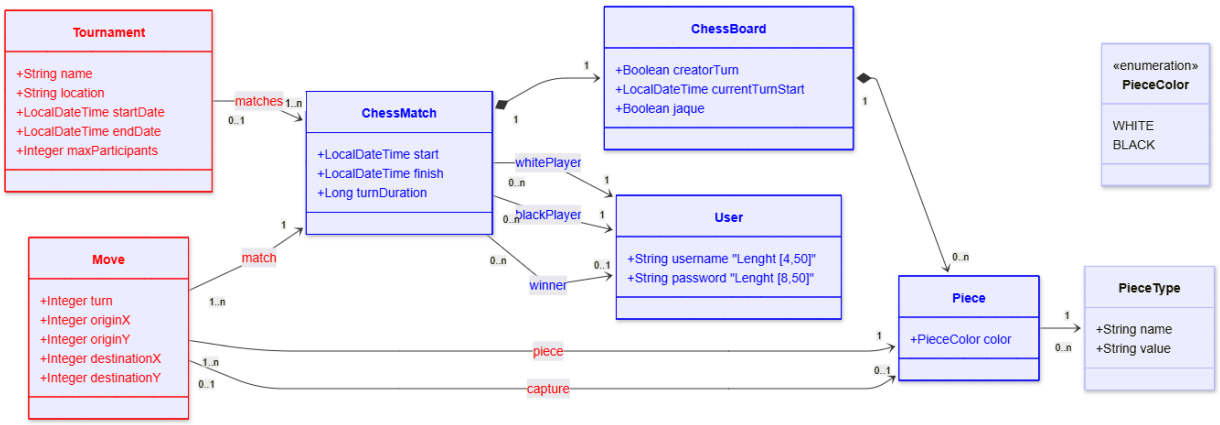
En este ejercicio añadiremos la funcionalidad de **gestión de partidas, jugadores y movimientos** dentro de una aplicación web de ajedrez.

Concretamente, se proporciona una clase *ChessMatch* que representa las partidas jugadas entre dos usuarios registrados en la plataforma. Cada partida se asocia a los jugadores que la disputan, y puede contener una lista de movimientos (*Move*) realizados durante la misma.

Además, se incluyen las clases *User* y *Tournament*, que representan a los usuarios del sistema y los torneos en los que participan, respectivamente.

Finalmente, se ha definido una relación que indica qué partidas pertenecen a un torneo concreto (llamada *matches*), lo que permitirá gestionar estadísticas por torneo.

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. Para comenzar esta prueba debe aceptar la tarea disponible en:

https://classroom.github.com/a/TMYDrant

Al aceptar dicha tarea, se creará un repositorio único individual para usted, debe usar dicho repositorio para realizar el control práctico. Debe entregar la actividad en EV asociada al control check

proporcionando como texto la dirección url de su repositorio personal. Recuerde que además debe entregar su solución del control.

La entrega de su solución al control se realizará mediante un único comando "*git push*" a su repositorio individual. 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 (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 dependen del 4) lo ejercicios del examen son independientes y puedes ser resueltos en cualquier orden.

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

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

- El ganador de las partidas (expresado a través de la propiedad *winner*) debe bien ser nulo (es decir la partida ha quedado en tablas/empate o aún no ha terminado), o si tiene valor, dicho jugador debe ser uno de los dos jugadores que han participado en ella (expresados a través de las propiedades *whitePlayer* y *blackPlayer*).
- 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 servicios de gestión de piezas, y del controlador de partidas

Parte 2A: Servicio de Gestión y respositorio (1 punto):

Modificar la clase "PieceService" y la interfaz "PieceRepository", para que sean un servicio Spring de lógica de negocio y un Repositiorio de Spring Data. Además se debe proporcionar una implementación a los métodos del servicio que use el repositorio:

Para el caso del servicio de gestión de Piezas debe permitirse:

1. Obtener todas las piezas de un color y tipo (método getPiecesByColorAndType(PieceColor color, String typeName)).
2. Obtener todas las piezas existentes (método getAll).
3. Obtener una pieza concreta por id (método getById).
4. Grabar una *pieza* en la base de datos (método *save*).

Todos estos métodos de la clase *PieceService* **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 partidas (1 punto):

Crear un método en el controlador "*ChessMatchController"* (alojado en el paquete us.es.dp1.chess.match) que permita devolver todas las partidas existentes. El método debe responder a peticiones tipo GET en la URL:

http://localhost:8080/api/v1/matches

Así mismo debe crear un método para devolver una partida concreta, este método debe responder a peticiones en la url http://localhost:8080/api/v1/matches/X donde X es la Id de la partida obtener, y devolverá los datos de la partida correspondiente. En caso de que no exista una partida 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 partidas 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 Partidas (2 puntos)

Modificar el componente React proporcionado en el fichero "frontend/src/matches/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/matches.

Si el conjunto de partidas no está vacío, este componente debe mostrar las partidas en una tabla (se recomienda usar el componente Table de reactstrap) incluyendo columnas para el nombre de la partida, sus fechas de inicio y fin (si lo hubiera), el nombre del jugador con blancas, el del jugador con negras, y el nombre del ganador. Los títulos de las cabeceras de las columnas de la tabla deben ser "Name", "Start", "Finish", "White", "Black", y "Winner" (y en ese orden).

Si el conjunto de partidas devuelto por la API es un array vacío, el componente debe mostrar un título de nivel 1 (h1), con el texto "NO MATCHES", 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 Move y Tournament y sus repositorios asociados

Parte 4A: Entidades y repositorios (1 punto)

Modificar las clases "*Move*" y "*Tournament*" para que sean entidades. Estas clases están alojadas en el paquete "*us.es.dp1.chess.game*", y deben tener los siguientes atributos y restricciones:

Para la clase Tournament:

- 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.
- Un 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.).
- Un atributo de tipo cadena de caracteres (*String*) llamado "location" 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 de tipo instante temporal (LocalDateTime) llamado "*startDate*" obligatorio, que representa el instante de comienzo del torneo.
- El atributo de tipo de tipo instante temporal (LocalDateTime) llamado "*endDate*", que representa el instante de finalización del torneo.
- El atributo de tipo entero (Integer) llamado "maxP*articipants* " que representa el máximo jugadores participantes en el torneo. Este *atributo es obligatorio* y tendrá un *valor mínimo de 2*.

Para la clase Move:

- 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 "turnNumber" (Integer) que es obligatorio y mayor o igual que 1, que representa el turno dentro de la partida en el que se realiza el movimiento.

- Atributo “originX” de tipo (Integer) que es obligatorio y toma valor entre 1 y 8, que representa la columna del tablero desde la que se mueve la pieza.
- Atributo “originY” de tipo (Integer) que es obligatorio y toma valor entre 1 y 8, que representa la fila del tablero desde la que se mueve la pieza.
- Atributo “destinationX” de tipo (Integer) que es obligatorio y toma valor entre 1 y 8, que representa la columna del tablero hacia la que se mueve la pieza.
- Atributo “destinationY” de tipo (Integer) que es obligatorio y toma valor entre 1 y 8, que representa la fila del tablero hacia la que se mueve la pieza.

Modificar las interfaces “MoveRepository” y “TournamentRepository” 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 “Tournament” hacia “ChessMatch” que exprese la que aparece en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades.

Además, se pide crear una relación unidireccional desde “Move” hacia “ChessMatch” que represente la que aparece en el diagrama UML, tenga en cuenta la cardinalidad que tiene, usando el atributo “match” en la clase “Move”. Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML, por ejemplo, algunos atributos pueden 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.

Finalmente, se pide crear las dos relaciones unidireccionales desde “Move” hacia “Piece” que representen las que aparece en el diagrama, usando como nombre de atributos “piece” y “capture”. Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML.

Test 5 – Modificación del script de inicialización de la base de datos para incluir dos Torneos, dos Movimientos, y sus relaciones

Modificar el script de inicialización de la base de datos, para que se creen los siguientes torneos (Tournament) y movimientos (Moves):

Tournament 1:

- id: 1
- name: “Abierto de Ajedrez Villa de Los Palacios y Vfca.”
- location: “Circulo Joven, Los Palacios y Villafranca”
- startDate: 10-11-2024 9:00:00
- endDate: 12-11-2024 17:00:00
- maxParticipants: 100

Tournament 2:

- id: 2
- name: “Torneo en Honor de José Maestre Amuedo”
- location: “Club cultural El Casino”

- startDate: 17-09-1990 17:00:00
- endDate: 20-09-1990 21:00:00
- maxParticipants: 100

Move 1:

- id: 1
- turn: 1
- originX: 4
- originY: 2
- destinationX: 4
- destinationY: 4

Move 2:

- id: 2
- turn: 7
- originX: 7
- originY: 5
- destinationX: 5
- destinationY: 6

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

- El *Tournament* cuyo id es 1 se asocie con las *partidas* con id 15
- El *Tournament* cuyo id es 2 se asocie con las *partida* con id 5 y 6
- El *Move* cuyo id es 1 tenga como partida asocia aquella cuyo id es 15.
- El *Move* cuyo id es 2 tenga como partida asocia aquella cuyo id es 5.
- El *Move* cuyo id es 1 tenga como pieza asocia aquella cuyo id es 131.
- El *Move* cuyo id es 2 tenga como pieza asocia aquella cuyo id es 5001.
- El *Move* cuyo id es 2 tenga como captura asociada la pieza cuyo id es 5002.

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.