Tarea online PROG03

Título de la tarea: Tarea online PROG03

Unidad: PROG03

Ciclo formativo y módulo: Desarrollo de Aplicaciones Multiplataforma (DAW). Programación

Curso académico: 2024/2025

¿Qué contenidos o resultados de aprendizaje trabajaremos?

Contenidos

PROG 03.- Utilización de objetos.

- Introducción.
- Fundamentos de la Programación Orientada a Objetos.
  - Conceptos.
  - Beneficios.
  - Características.
  - Lenguajes de programación orientados a objetos.
- Clases y Objetos. Características de los objetos.
  - Propiedades y métodos de los objetos.
  - Interacción entre objetos.
  - Clases.
- Utilización de objetos.
  - Ciclo de vida de los objetos.
  - Declaración.
  - Instanciación.
    - Concepto de constructor.
  - Referencias a objetos.
  - Manipulación.
  - Destrucción de objetos y liberación de memoria.
  - Objetos String en Java.
- Métodos.
  - Parámetros y valores devueltos.
  - Uso de métodos.
  - Documentación de una clase.
  - Objetos inmutables.
  - Comparación de objetos en Java: método equals.
  - Métodos estáticos.
    - Ejemplos: generar números aleatorios.
    - Métodos "fábrica" o pseudoconstructores.
  - Clases envoltorio en Java.

Representación textual de un objeto en Java: método toString.
Bibliotecas de clases o librerías de objetos (paquetes).
Sentencia **import**.
Bibliotecas Java.
Programación de la consola: entrada y salida de la información.
Conceptos sobre la clase **System**.
Entrada por teclado. Clase **System**.
Entrada por teclado. Clase **Scanner**.
Salida por pantalla.
Salida de error.
Manipulación de la fecha y la hora en Java.
LocalDate.
LocalTime.
LocalDateTime.
Formateado de fechas.
ChronoUnit.
Excepciones.
Capturar una excepción con try.
El manejo de excepciones con catch.

Resultados de aprendizaje

- **RA2**. Escribe y prueba programas sencillos, reconociendo y aplicando los fundamentos de la programación orientada a objetos.

1.- Descripción de la tarea

Caso práctico

En **BK Programación** tienen un nuevo encargo: han de desarrollar una nueva versión del popular juego de las damas. La parte gráfica la desarrollarán más adelante. De momento, **Ada** ha encargado a **María** que comience a definir unas clases básicas que van a usar para el programa. En concreto un par de clases llamadas "Dama" y "Posicion".

Juan, que ha visto a **María** muy nerviosa tras el encargo de **Ada**, se ofrece a echarle una mano:

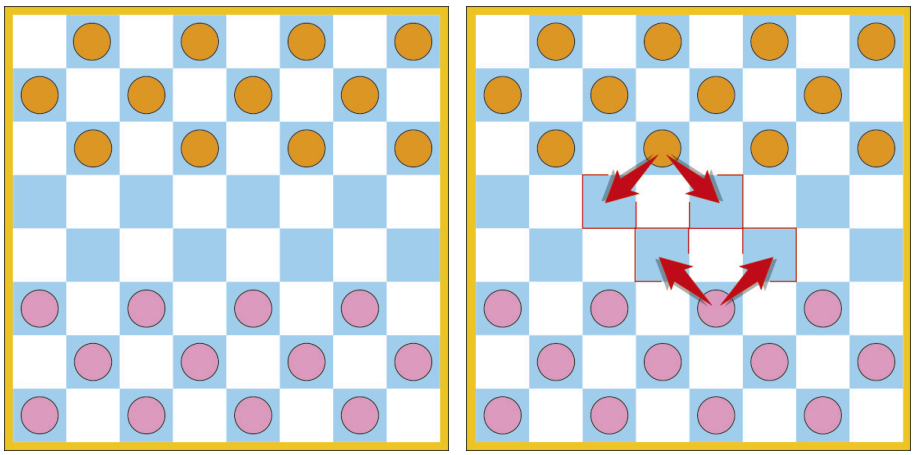
—**María**, no te preocupes, lo que tienes que hacer es muy fácil, son unas clases muy sencillas con unos pocos métodos para crear el objeto, mover, ¡y poquita cosa más!

—Pues yo esto lo veo muy negro,… — piensa **María** para sí.

—Venga, no te agobies, vamos a echar un café y nos ponemos manos a la obra,… —insiste **Juan**.

¿Qué te pedimos que hagas?

La tarea va a consistir en modelar el movimiento de una dama por el tablero de dicho juego. **Para controlar la posición en de la dama en las casillas del tablero, se utilizarán las coordenadas que vienen dadas en el tablero de ajedrez, por una fila (del "1" al "8") y una columna (de la "a" a la "h").**

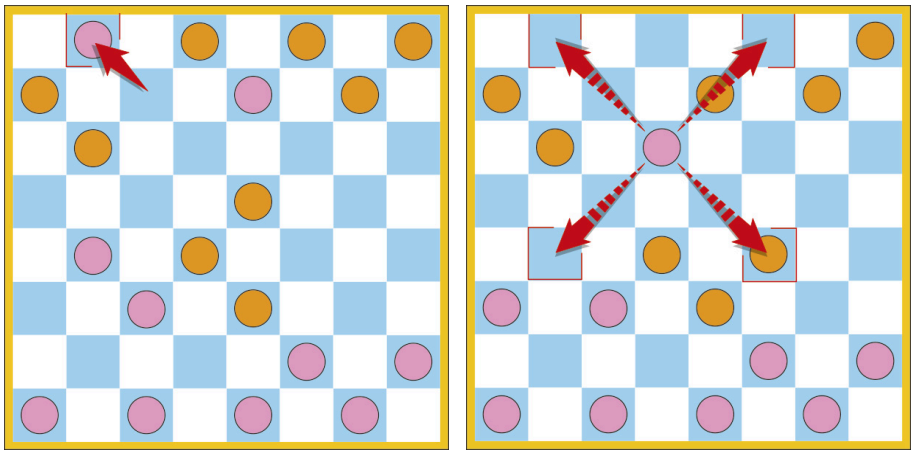


Imágenes obtenidas de la web: www.hagaloustedmismo.cl

En la primera imagen puedes observar cuál es el posicionamiento inicial válido para las damas dependiendo de su color. En la segunda imagen puedes apreciar cuáles son **los movimientos válidos para una dama, siempre hacia adelante, en diagonal y solamente una casilla**.

Deberás controlar que el movimiento de una dama se realice correctamente siguiendo las reglas descritas anteriormente y además no estará permitido realizar movimientos que puedan sacar a la pieza fuera del tablero.

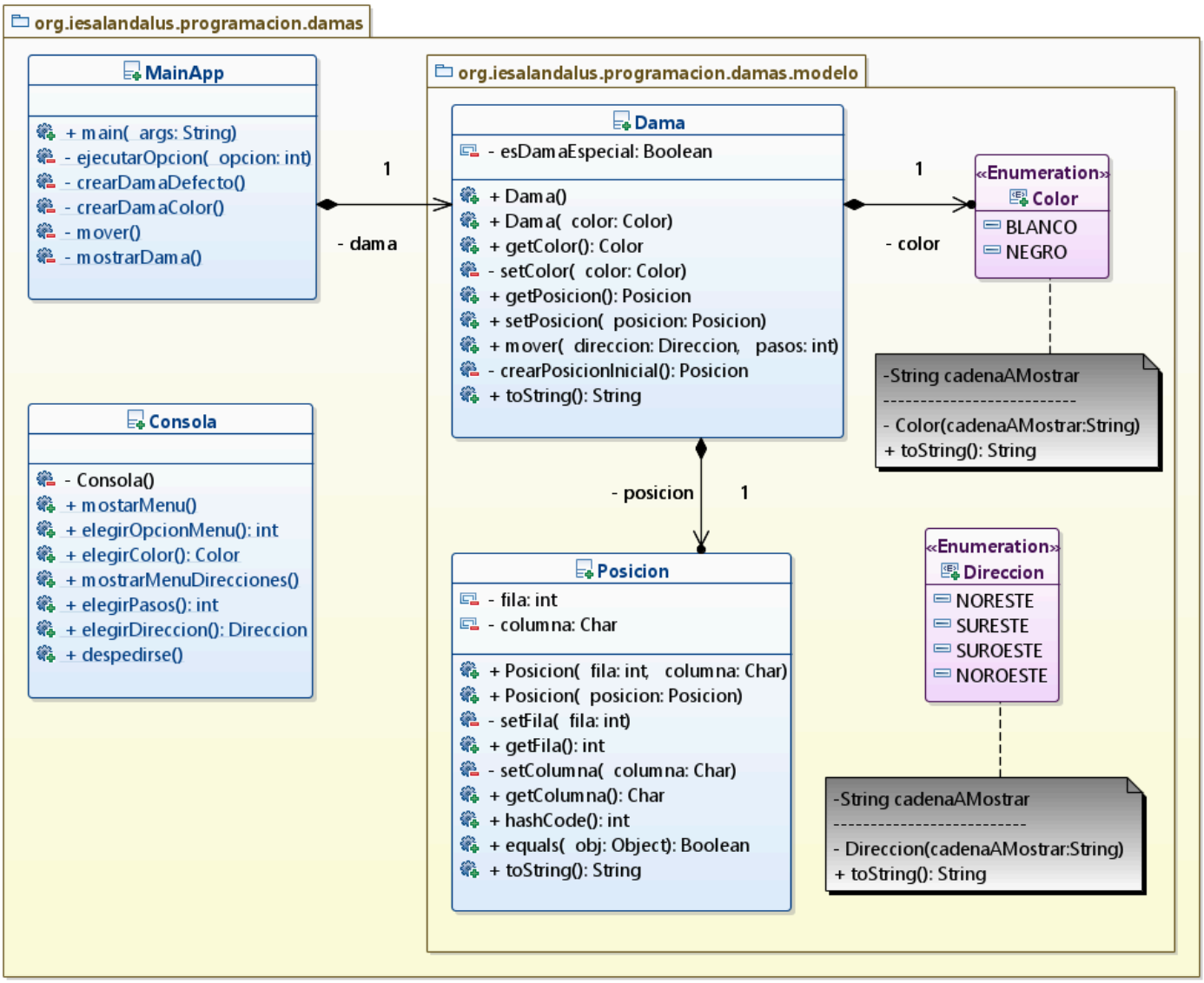
Además, **la dama adquiere unos movimientos especiales cuando consigue llegar el extremo contrario del tablero.** En ese momento, la dama puede moverse hacia adelante y hacia atrás en el tablero, siempre en diagonal y todas las casillas que desee, siempre y cuando no se salga del tablero.



Imágenes obtenidas de la web: www.hagaloustedmismo.cl

He subido a GitHub un esqueleto de proyecto gradle que ya lleva incluidos todos los test necesarios que el programa debe pasar y las dependencias, entre ellas la de la librería **Entrada**. La URL del repositorio es la siguiente: repositorio GitHub con el esqueleto del proyecto.

Además, te pongo un diagrama de clases para ayudarte y poco a poco te iré explicando los diferentes pasos a seguir:



Primeros Pasos

Lo primero que debes hacer es un **fork** del repositorio donde he colocado el esqueleto de este proyecto.
Clona tu repositorio remoto recién copiado en GitHub a un repositorio local que será donde irás realizando lo que a continuación se te pide. Modifica el archivo **README.md** para que incluya tu nombre en el apartado "Alumno". Realiza tu primer commit.

Enumerado Direccion (CE 2.B)

Crea un enumerado llamado **Direccion** que contenga los literales: **NORESTE**, **SURESTE**, **SUROESTE**, **NOROESTE**. Los literales estarán parametrizados y a cada uno le pasaremos la representación en cadena de los mismos: "Noreste", "Sureste", "Noroeste" o "Suroeste". Realiza un commit.
Crea el atributo cadenaAMostrar (String). Realiza un commit.
Crea el constructor con la visibilidad adecuada, que almacenará en el atributo anterior la cadena que se le pase por parámetro. Realiza un commit.
Crea el método toString que devolverá la representación en forma de cadena del literal. Realiza un commit.

Enumerado Color (CE 2.B)

Crea un enumerado llamado **Color** que contenga los literales: **BLANCO** y **NEGRO**. Los literales estarán parametrizados y a cada uno le pasaremos la representación en cadena de los mismos: Blanco o Negro. Realiza un commit.
Crea el atributo **cadenaAMostrar** (String). Realiza un commit.

Crea el **constructor** con la visibilidad adecuada, que almacenará en el atributo anterior la cadena que se le pase por parámetro. Realiza un commit.
Crea el método **toString** que devolverá la representación en forma de cadena del literal. Realiza un commit.

Clase Posicion (CE 2.D)

Crea la clase **Posicion**. Crea los atributos **fila** (int) y **columna** (char) con la visibilidad adecuada. Realiza un commit.
Crea los métodos **get** y **set** para los atributos. Recuerda que para el método **set** se debe tener en cuenta que los valores indicados sean correctos **(las filas van del 1 al 8 -ambos inclusive- y las columnas de la 'a' a la 'h' -ambas inclusive-)** y si no, que se lance una excepción del tipo **IllegalArgumentException** con el mensaje adecuado. En caso de que la posición no sea la correcta no deben modificarse los atributos de la dama. Realiza un commit.
Crea un constructor para esta clase que acepte como parámetros la **fila** y la **columna** y que los asigne a los atributos si son correctos. Si no son correctos debe lanzar una excepción del tipo **IllegalArgumentException** con el mensaje adecuado. Para ello utiliza los métodos **set** anteriormente creados. Realiza un commit.
Crea el constructor copia para esta clase. Realiza un commit.
Crea los métodos **equals** y **hashCode** para esta clase. Realiza un commit.
Crea el método **toString** que devolverá un **String** y será la representación de la fila y la columna de forma adecuada (**fila=valorFila, columna=valorColumna**). Realiza un commit.

Clase Dama (CE 2.D)

Crea la clase **Dama** cuyos atributos serán un **color** (del tipo enumerado **Color**), una **posicion** (de la clase **Posicion**) y un **esDamaEspecial** (del tipo booleano), con la visibilidad adecuada. Realiza un commit.
Crea los métodos **get** y **set** para cada atributo con la visibilidad adecuada. Los métodos **set** siempre comprobarán la validez de los parámetros pasados y si no son correctos deberá lanzar la excepción adecuada con el mensaje adecuado. Realiza un commit.
Crea un constructor por defecto para esta clase que cree una dama blanca en la posición correcta y aleatoria del tablero. La posición inicial deberá estar en alguna casilla negra dentro de las filas 1, 2 o 3. Realiza un commit.
Crea un constructor para la clase que acepte como parámetro el color y que creará una dama de dicho color cuya posición inicial estará en una casilla negra de alguna de las filas 1,2 o 3 (si es blanca) o en alguna de las filas 6, 7 y 8 (si es negra). Realiza un commit.
- Para obtener la posición inicial de manera aleatoria, implementa el método **crearPosicionInicial.**

Crea el método **mover** que acepte como parámetros una **Direccion y el número de pasos que se moverá.** Deberás tener en cuenta las siguientes restricciones: Realiza un commit.
- La dirección no puede ser nula o de lo contrario debe lanzar una excepción adecuada (**NullPointerException** o **IllegalArgumentException**) con el mensaje adecuado.
  - Si la dama todavía no se ha convertido en dama especial, solamente podrá moverse en un dirección que le permita avanzar en el tablero y nunca retroceder. **Noreste o Noroeste (si es una dama blanca) y Sureste o Suroeste (si es una dama negra)**.
- El número de casillas que se mueve la dama deberá ser un número entero positivo, mayor o igual que 1.
  - Si la dama todavía no se ha convertido en dama especial, el numero de casillas que se mueve será 1.
- Si la dama llega al extremo del tablero (fila 8 si es blanca o fila 1 si es negra) se convertirá en dama especial modificando el atributo **esDamaEspecial** y poniéndolo al verdadero.
- Si no puede realizar dicho movimiento, debido a que la dama se sale del tablero o que no está permitido porque todavía no es una dama especial, se debe lanzar una excepción del tipo **OperationNotSupportedException** con un mensaje adecuado y no

modificarán los atributos de la dama. Si el movimiento es válido, se modificará la posición actual de la dama.

Crea el método `toString` que devuelva un `String` que será la representación de dicho objeto (color y posición). Realiza un commit.

Clase Consola (CE 2.E)

Crea la clase de utilidades `Consola`. Realiza un commit.
Crea el constructor para esta clase con su visibilidad adecuada, teniendo en cuenta que será una clase de utilidades que sólo contendrá métodos estáticos. Realiza un commit.
Crea el método `mostrarMenu` que mostrará el menú con las opciones de nuestra aplicación: crear dama por defecto, crear dama eligiendo el color, mover y salir. Realiza un commit.
Crea el método `elegirOpcionMenu` que mostrará un mensaje para que elijamos una opción del menú anteriormente creado y nos pedirá que introduzcamos por teclado la opción hasta que ésta sea valida. Devolverá la opción elegida. Realiza un commit.
Crea el método `elegirOpcion` que nos pedirá que elijamos un color mientras éste no sea válido y dependiendo de la opción elegida devolverá un color u otro. Realiza un commit.
Crea el método `mostrarMenuDirecciones` que mostrará por consola un menú con las diferentes direcciones que podemos elegir. Realiza un commit.
Crea el método `elegirDireccion` que mostrará un mensaje indicando que elijamos una dirección del menú anteriormente creado y nos pedirá que introduzcamos por teclado la opción hasta que ésta sea valida. Devolverá la dirección elegida. Realiza un commit.
Crea el método `elegirPasos` que nos pedirá que introduzcamos por teclado el número de casillas, el cual deberá ser un número entero positivo mayor o igual que 1.
Crea el método `despedirse` que mostrará un mensaje de despedida al salir de nuestra aplicación.

Clase MainApp (CE 2.G)

Crea el atributo de clase `dama`. Realiza un commit.
Crea el método `ejecutarOpcion` que dependiendo de la opción pasada como parámetro, actuará en consecuencia. Realiza un commit.
Crea el método `crearDamaDefecto` que asignará al atributo de clase `dama` una nueva instancia de una dama creada con el constructor por defecto. Realiza un commit.
Crea el método `crearDamaColor` que asignará al atributo de clase `dama` una nueva instancia de una dama creado con el constructor al que le pasamos el color. Realiza un commit.
Crea el método `mover` que mostrará un menú con las posibles direcciones. Nos preguntará por la dirección a mover. Si la dama es especial, nos también preguntará el número de casillas que queremos mover. Moverá la dama a la nueva posición, si es posible. Realiza un commit.
Crea el método `mostrarDama` que nos mostrará la información de la dama (color y posición) si ésta está creado. De lo contrario nos informará de ello.
Crea el método `main` que será el método principal de nuestra aplicación y deberá iterar mostrando el menú principal, pidiendo la opción y ejecutándola mientras no elijamos salir, en cuyo caso mostrará un mensaje de despedida y nuestra aplicación finalizará. Realiza un `commit` y realiza el `push` a tu repositorio remoto en GitHub.

Se valorará (CE 2.I):

- La indentación debe ser correcta en cada uno de los apartados.
- Los identificadores utilizados deben ser adecuados y descriptivos.
- Se debe utilizar la clase `Entrada` para realizar la entrada por teclado que se encuentra como dependencia de nuestro proyecto en la librería `entrada`.
- El programa debe pasar todas las pruebas que van en el esqueleto del proyecto y toda entrada del programa será validada, para evitar que el programa termine abruptamente debido a una excepción.
- La corrección ortográfica tanto en los comentarios como en los mensajes que se muestren al usuario.

- **Para calificar cada uno de los criterios de evaluación asociados a la tarea será imprescindible que el código compile correctamente y se pueda ejecutar. En caso contrario, los criterios de evaluación serán calificados con un 0.**

2.- Información de interés

Recursos necesarios y recomendaciones

- Un IDE de desarrollo (NetBeans o Eclipse). En dicho IDE deberás haber instalado el plugins que permita trabajar con proyectos gradle.
- La URL del repositorios es la siguiente: repositorio GitHub con el esqueleto del proyecto.

Indicaciones de entrega

Deberás crear un archivo de texto que incluya las explicaciones sobre los puntos que las merezcan según tu criterio. Es imprescindible que en este documento incluyas la URL al repositorio GitHub que has debido crear para realizar esta tarea.

Empaquetarás en un archivo comprimido el documento de texto anterior y el proyecto creado con el IDE que hayas utilizado.

El envío del archivo comprimido anterior lo realizarás a través de la plataforma. El archivo se nombrará siguiendo las siguientes pautas:

Apellido1_Apellido2_Nombre_PROG_Tarea03

3.- Evaluación de la tarea

Criterios de evaluación implicados

Criterios de Evaluación RA2

- b.- Se han escrito programas simples.
- d.- Se han utilizado métodos y propiedades de los objetos.
- e.- Se han escrito llamadas a métodos estáticos.
- g.- Se han incorporado y utilizado librerías de objetos.
- i.- Se ha utilizado el entorno integrado de desarrollo en la creación y compilación de programas simples.

¿Cómo valoramos y puntuamos tu tarea?

Rúbrica de la tarea		
Criterios de Evaluación	**Calificación**	**Retroalimentación**
2b		
2d		
2e		
2g		
2i		