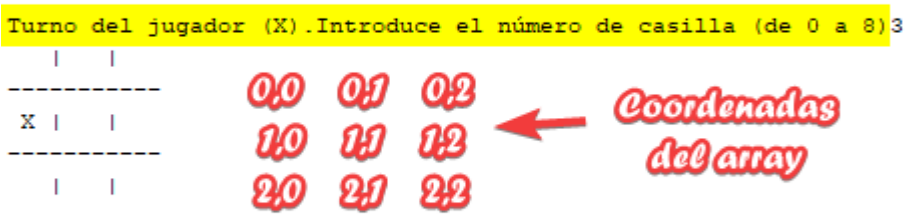
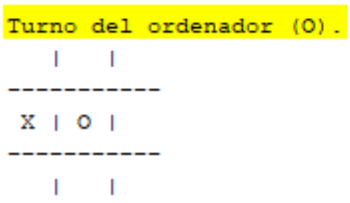
Actividades de repaso

Ejercicio 1. Tres en raya.

Se trata de realizar un sencillo juego de tres en raya para jugar contra el computador, con la estrategia muy básica.

Para ello tendremos una enumeración: `TipoFicha` y tres clases: `Ficha`, `Tablero` y `Game`. La enumeración ***TipoFicha*** va a poder ser *X* (`CRUZ`), *O* (`CIRCULO`) o vacía (`VACIA`). La ***Ficha*** tiene, únicamente, un atributo de tipo de ficha. El ***Tablero*** es un array de 3x3 de tipo Ficha. El ***Juego*** es la implementación del juego.





Enumeración ***TipoFicha***

Esta enumeración representa el tipo de fichas del juego con 3 posibles valores: `CRUZ`, `CIRCULO` o `VACIA`.

Atributo (final)

- Atributo `final` denominado `caracter` con el carácter asociado a cada tipo *'X'*, *'O'* y *' '*, respectivamente.

Constructor

- Un constructor privado que consideres oportuno para asignarle el carácter.

Métodos

Un método que devuelva el carácter, *getCaracter()*, y un método *toString()* que devuelva el carácter como cadena, tipo *"X"*, *"O"* y *" "* (convierte el carácter a cadena, **no uses un if/switch**).

Clase ***Ficha*** implementa la interface ***Comparable<Ficha>***

Atributo (final)

Un único atributo `final`, denominado *tipo*, de *TipoFicha* con el tipo de ficha, `CRUZ`, `CIRCULO` o `VACIA`.

Constructores

- Un constructor **por defecto** que crea una ficha con tipo `VACIA`.
- Un constructor que **recoge el tipo** y se lo asigna al atributo.

Métodos

- *isEmpty():* devuelve verdadero si el tipo de la ficha es nula o es `VACIA`.
- *toString*: devuelve la representación del tipo de la ficha como cadena (el mismo de la enumeración *TipoFicha*). Invoca al *toString()* de tipo.
- *equals()* y *hashCode()*: sobrescribe los métodos de *Object* de modo que devuelva que dos objetos Ficha son iguales si tiene el mismo valor de *tipo*. El método *hashCode*() tiene que devolver, al menos, el mismo valor cuando dos fichas son del mismo tipo.
- *compareTo*: devuelve 0 cuando las fichas son del mismo tipo, 1 cuando la ficha es de tipo CRUZ y -1 cuando la ficha es de tipo CIRCULO, estas dos condiciones independientemente del tipo de la ficha recogida.

Clase ***Tablero***

Contiene el tablero del juego, un **array de dos dimensiones de 3x3** con las fichas. Puedes crear una constante TAMANHO con el valor 3 para facilitar la codificación.

Atributo (final)

- *fichas*: array (*final*) de **dos dimensiones 3x3 de objetos de tipo *Ficha***.

Constructor

- Un único **constructor por defecto** que crea un Tablero de 3x3 fichas. Mediante un doble bucle, crea cada ficha con el constructor por defecto de *Ficha*.

Métodos

- *addFicha*: recoge las coordenadas **fila**, **columna** y la **ficha** e intenta situarla en esa posición (i, j). Devolviendo si ha podido añadir la ficha.

  Debes comprobar que **los índices de fila y columna están dentro del rango** de posibles valores y que la casilla está vacía. Si no es así devuelve `false`. Si los valores de los índices son correctos, la pone en esa posición y devuelve `true`.
- *isEmpty*: recoge las coordenadas de la fila y columna y devuelve `true` si la ficha es nula o está vacía, `false` en caso contrario.
- *isFull*: no recoge nada y devuelve `false` cuando alguna ficha del tablero está vacía. Verdadero en caso contrario. *Ayuda: recorre todo el tablero y devuelve falso cuando encuentra una vacía. Al final, verdadero.*
- *isWinner*: recoge **una ficha** y mira si hay tres en raya en el tablero para esa ficha, devolviendo **verdadero si hay tres en raya**.

  Esto es: **recorre cada fila** y mira si las tres fichas de cada fila son iguales (método *equals*) a la ficha recogida; **recorre las columnas** y mira si las tres fichas de cada fila son iguales a la ficha; **comprueba** si alguna de **las dos diagonales** tiene las tres fichas iguales a la ficha recogida. Debe devolver `true` en cuanto encuentra un 3 en raya.
- Sobrescribe *toString* para que devuelva el tablero como cadena con la siguiente representación (dependiendo de los valores de las fichas). **Sólo se considerará correcto si se emplea *StringBuilder***.

```
 O | X | O
-----------
   | X |
-----------
   |   |
```

Clase ***Game***

Representa al Juego de tres en raya. Por ello, tiene: un *tablero*, **el tipo de ficha del jugador actual:** *tipoJugador*, y, además, precisamos leer desde teclado con Scanner.

Atributos

- *tablero*: de tipo *Tablero*, representa al tablero del juego.
- *tipoJugador*: de tipo *TipoFicha*, el tipo de ficha del jugador actual (CRUZ o CIRCULO, dependiendo del jugador actual).
- *lector*: de tipo Scanner, para leer de teclado.

Constructor

- Un único **constructor por defecto** que crea un tablero con el constructor por defecto, asigna a *tipoJugador* un valor aleatorio entre los dos posibles valores y crea un ***Scanner*** para leer de teclado. Recuerda que existe una clase para generar números aleatorios: `Random r = new Random()`, sobre el que puedes invocar el método `r.nextInt(2)` para que devuelva 0 o 1, eligiendo CRUZ o CICULO dependiendo de ese valor.

Métodos

- *getCoordenadas*: método **estático** que recoge un número, de 0 a 8, que identifica una casilla, y devuelve las coordenadas asociadas a la posición de ese número dentro del tablero (ved imagen inicial). Debe devolver un objeto de tipo Point con las coordenadas. *En la siguiente imagen aparecen los posibles valores de entrada:*

```
 0 | 1 | 2
-----------
 3 | 4 | 5
-----------
 6 | 7 | 8
```

  Así, por ejemplo, si recoge un 8 devuelve el punto *2, 2*. Si recoge un 4 devuelve el punto 1 ,1; si recoge un 3 devuelve 1,0; etc. **La coordenada de la fila devuelta se corresponde con el índice/3 y la coordenada de la columna con el módulo: índice%3.**
- *doComputerMovement*: método que no recoge nada y sitúa la ficha del computador en la primera casilla vacía de este *array*: {4, 0, 2, 6, 8, 1, 3, 5, 7} (*Existen algoritmos mejores, por supuesto, pero sólo es para que demostréis que sabéis recorrer bucles y crear arrays estáticos*).

- *play*: implantación del juego:
  - Muestra el tablero y **va pidiendo la introducción de la posición de la ficha de manera alternativa al jugador y al computador**, según le toque.
  - **Si el tablero está lleno**, método *isFull*, sale del bucle.
  - **Si le toca al jugador**, pide el índice dentro del tablero y añade la ficha a esa posición si es una posición válida.
  - **Si le toca al computador**, sitúa la ficha por medio del método *doComputerMovement*.
  - **Cambia el turno al final de cada iteración**.
  - Si el tablero está lleno o existe un ganador, termina la partida.

```
¡Bienvenido al juego del 3 en raya!
   |   |
-----------
   |   |
-----------
   |   |


Turno del ordenador (O).
   |   |
-----------
   | O |
-----------
   |   |


Turno del jugador (X).Introduce el número de casilla (de 0 a 8): 5
   |   |
-----------
   | O | X
-----------
   |   |


Turno del ordenador (O).
 O |   |
-----------
   | O | X
-----------
   |   |
```

- *main*: Crea un juego e invoca al método ***play***.

Ejercicio 2. Programa de gestión de exámenes

Se desea realizar una aplicación para gestión de exámenes, que pueden contener preguntas de tipo cuestión o tipo test.

Clase ***Opcion***

Representa cada una de las opciones de una pregunta tipo test. Tiene:

Atributos (finales)

- ***enunciado***: texto (*final*) con el enunciado de la opción de la pregunta tipo test.
- ***correcta***: booleano (*final*) que indica si es una opción correcta o no.

Constructor

Un único constructor que recoge el ***enunciado*** y si es ***correcta*** o no.

Métodos

- ***getEnunciado***: devuelve el enunciado.
- ***isCorrecta***: devuelve si es correcta o no.
- Sobrescribe ***toString*** para que devuelva el enunciado. Si la opción es correcta devuelve el enunciado con un [*] al final de la cadena.

Clase abstracta ***Pregunta*** implementa la interface ***Comparable<Pregunta>***

Pregunta es una clase **abstracta**, que implementa la interface ***Comparable***<*Pregunta*>, con los atributos y comportamientos comunes a todas las preguntas:

Constante

DEFAULT_VALUE: **1**, puntos y número de pregunta por defecto al asignar valore son válidos o al crear el objeto.

Atributos

- Identificador de la pregunta, de tipo Integer: *idPregunta*.
- El texto del enunciado (*enunciado*), como cadena. De tipo *final*.
- Descripción de la pregunta (*descripcion*), como cadena.
- Número de pregunta, *numero*, de tipo entero.
- Puntos que vale la pregunta, *puntos*, de tipo double.

Constructor

Un único constructor que recoge el ***enunciado*** y el ***número***. Debe comprobar que el número es mayor o igual a 1; si no lo es, pone el valor por defecto. Además el atributo puntos al valor por defecto.

Métodos

- De tipo ***get*** y ***set*** para todos los atributos, exceptuando el método ***setEnunciado***, pues el atributo enunciado es final y no debe tener método ***set***. A ser posible, **los métodos *set* deben devolver una referencia al propio objeto** para poder concatenar las asignaciones.

  El método *setPuntos* debe comprobar que los puntos son mayores que 0, dando el valor por defecto si no lo es.

  El método *setNumero* debe comprobar que el número es mayor que 0, dando el valor por defecto si no lo es.

- Método ***toString*** que devuelve el número y el enunciado de la pregunta. Con el siguiente formato: *número. enunciado:*

5. Supongamos que tiene una colección de productos a la venta en una base de datos

… Si el enunciado es *null* debe devolver un guion en vez de escribir null.

- Implementación del método ***compareTo*** de la interface ***Comparable<Pregunta>***, de modo que **compare por número de pregunta**.

Clase ***PreguntaTest*** hereda de ***Pregunta*** e implanta ***Predicate<Integer>***

La interface ***java.util.function.Predicate*** es una interface funcional, tiene un único método, ***test***, que recoge un tipo de dato y lo valida, devolviendo verdadero o falso si cumple la condición que se implante. En este caso, se trata de implantar el método *public boolean test(Integer dato)*.

Las preguntas tipo test tiene únicamente un **array de opciones**, de tipo *Opcion*, de la pregunta.

Constante

- *NUMERO_OPCIONES*: 4, representa el número de opciones por defecto.

Atributo

- *opciones*: atributo final de tipo array de ***Opcion***.

Constructores

Dos constructores:

- Uno que recoge el *enunciado* y el **número de pregunta** (*numero*), creando el array de opciones con el número de opciones por defecto.
- Uno que recoge el *enunciado*, el **número de pregunta** y el **número de opciones**, creando un *array* con esas opciones.

Métodos

- ***getOpciones***: devuelve las opciones.
- ***addOpcion***: recoge una opción (de tipo *Opcion*) y la añade, recorriendo el array en busca de un espacio *null*. Devuelve si la ha podido añadir o no.
- ***getNumCorrectas***: devuelve el número de opciones correctas de la pregunta. Recorre las opciones y cuenta las correctas.
- ***getPuntos****(int[] marcadas)*: recoge un array de enteros con los números de las opciones marcadas (puedes marcar varias) y devuelve los puntos obtenidos. Si el array de enteros recogido es nulo debe devolver 0. Las incorrectas cuentan negativo. Por ello:

  Para ello, debes recorrer el *array* de recogido (*marcadas*) y comprobar si es correcta o no, llevando cuenta de las correctas y las incorrectas.

  Los puntos se calculan con las formula:

$$puntos = \frac{puntos\ de\ la\ pregunta * (marcadas\ bien - marcadas\ mal)}{numero\ opciones\ correctas\ de\ la\ pregunta}$$

- ***toString***: devuelve el enunciado (invoca al *toString* de la clase padre) y la lista de opciones con el número de opción:

7. ¿Cuál de estas declaraciones compila? (Elija todas las opciones que correspondan).
[1] A. HashSet<Number> hs = new HashSet<Integer>();
[2] B. HashSet<? super ClassCastException> set = new HashSet<Exception>();[*]
[3] C. List<String> list = new Vector<String>();[*]
[4] D. List<Object> values = new HashSet<Object>();

Importante: emplea la clase StringBuilder para crear la cadena. Debemos conocer a la perfección dicha clase.

- ***test***: implantación del método *test* de la interface. Recoge el Integer y devuelve verdadero si la opción seleccionada es correcta. Comprueba que el valor recogido es un valor válido entre 0 y el número de opciones, además de comprobar que esa opción no es nula.

Clase ***Examen***

Atributos

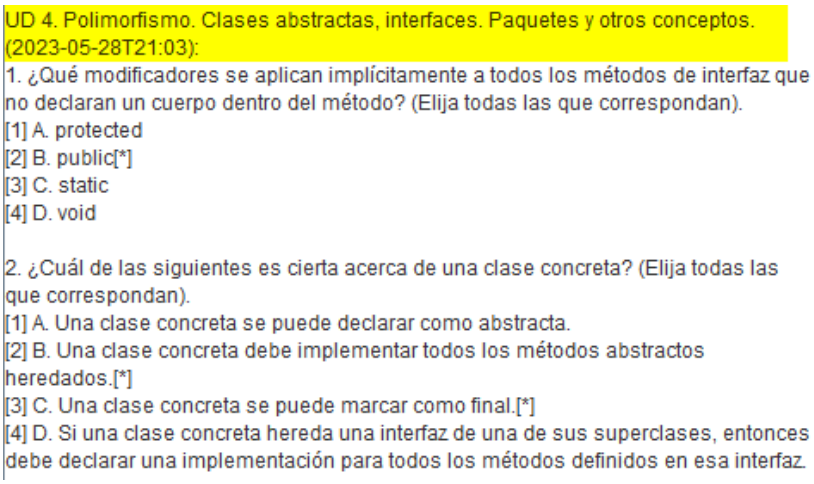
- *idExamen*: de tipo *Integer*, identificador del examen.
- *preguntas*: List de preguntas del examen.
- *descripcion*: nombre del examen.
- *fecha*: de tipo *LocalDateTime*, con la fecha del examen.

Constructores

- Un constructor que recoge la **descripción**, crea la lista de preguntas y pone al fecha como la actual (*ahora()*).
- Un constructor que recoge la **descripción**, el **año**, **mes**, **día**, **hora**, **minuto** (todos enteros), creando la fecha con esos valores (recuerda el método estático para hacerlo). Además, crea la lista de preguntas.

Métodos

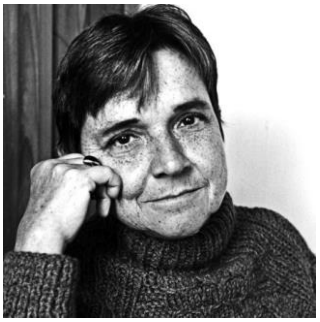
- Get/set para cada atributo.
- ***addPreguntas***: recoge una lista de preguntas y las añade al examen.
- ***removePregunta***: recoge una pregunta y la borra.
- ***ordenar***: ordena la lista de preguntas.
- ***equals()*** y ***hashCode()***: sobrescribe los métodos de *Object* de modo que devuelva que dos objetos son iguales si tiene el mismo valor de *idExamen*. El método hashCode() tiene que devolver, al menos, el mismo valor cuando dos exámenes son iguales.
- ***toString()***: sobrescribe el método de *Object* de modo que devuelva el examen. La descripción, la fecha entre paréntesis y la lista de preguntas. Algo así:


UD 4. Polimorfismo. Clases abstractas, interfaces. Paquetes y otros conceptos. (2023-05-28T21:03):
1. ¿Qué modificadores se aplican implícitamente a todos los métodos de interfaz que no declaran un cuerpo dentro del método? (Elija todas las que correspondan).
[1] A. protected
[2] B. public[*]
[3] C. static
[4] D. void

2. ¿Cuál de las siguientes es cierta acerca de una clase concreta? (Elija todas las que correspondan).
[1] A. Una clase concreta se puede declarar como abstracta.
[2] B. Una clase concreta debe implementar todos los métodos abstractos heredados.[*]
[3] C. Una clase concreta se puede marcar como final.[*]
[4] D. Si una clase concreta hereda una interfaz de una de sus superclases, entonces debe declarar una implementación para todos los métodos definidos en esa interfaz.

Importante: emplea la clase StringBuilder para crear la cadena.

Epílogo. Un poema de Adrienne Rich



Adrienne Cecile Rich (16 de mayo de 1929, Baltimore, Maryland - 27 de marzo de 2012, Santa Cruz, California), más conocida como Adrienne Rich, fue poeta, ante todo, pero también ensayista y académica. Publica su primera colección con poco más de veinte años. En 1953 se casa con un profesor de economía con el que tiene tres hijos. Continúa escribiendo poesía mientras cría en el contexto de la Guerra Fría. Poco después de separarse de su marido, él se suicida. Todas esas experiencias se vuelcan de un modo u otro en su trabajo, ya sea poético o ensayístico, especialmente en *Nacemos de mujer*. A comienzos de la década de 1970, Rich se involucra en el activismo por los derechos civiles y contra la Guerra de Vietnam. Además de escribir sobre la maternidad desde el feminismo, impulsa la visibilidad lésbica, sin parar de investigar sobre el lenguaje. Falleció en marzo de 2012, dejando una prolongada carrera de escritora y algunas de las piezas más hermosas de la poesía del siglo XX en lengua inglesa.

VII

De pronto ya no me parece
viable este mundo:
tú estás ahí fuera quemando las cosechas
con un nuevo sublimado
Esta mañana dejaste el lecho
que aún compartimos
y saliste a esparcir impotencia
por el mundo

Te odio.
Odio la máscara que llevas, tus ojos
que fingen una profundidad
que no poseen, que me arrastran
hasta el antro de tu cráneo
el paisaje de osamenta
odio tus palabras
me hacen pensar en falsos
bonos revolucionarios
crujiente imitación de pergaminos
en venta en los campos de batalla.

Anoche, en este cuarto, llorando
te pregunté: ¿Qué sientes tú?
¿Sientes algo?

Ahora, en la contorsión de tu cuerpo,
mientras defolias los campos que nos sustentaban
tengo tu respuesta.