PLAN DE REFUERZO
ACTIVIDADES 2º TRIMESTRE

SEMANA 4-10 de MAYO

EJERCICIO 1

Implementar la siguientes clases decidiendo los tipos de datos correctos, con lo métodos solicitados e implementando los métodos como consideres oportuno (puede ser simplemente un mostrar un mensaje). Reflexiona y repasa los conceptos Clase, Objeto, Atributos, Visibilidad de Atributos.

Deberéis explicar en los comentarios todas las decisiones tomadas.

Clase Avión:

- Propiedades: Aerolínea, Longitud, Modelo, ModoVuelo que podrá tomar los siguiente valores (MANUAL, CRUCERO, PILOTO_AUTOMÁTICO)
- Comportamiento: despegar(), aterrizar(), volar()

Clase Pasajero:

- Propiedades: Nombre,Dirección y número de pasaporte y estado (EMBARCADO, PENDIENTE)
- Comportamiento: embarcar() , sonreir() , andar()

Además de los métodos solicitados deberéis implementar los constructores que consideréis oportunos, los getter/setter, equals y el método toString() para ambas clases.

Crear en un Main.java dos objetos de la clase avión y 5 objetos de la clase Pasajero. Utilizar para cada objeto algún método de los implementados.

EJERCICIO 2

Modificar las clases del ejercicio uno añadiendo l**os atributos y métodos de clase** necesarios para controlar en todo momento el número de aviones y pasajeros que se han creado.

Demostrar la utilización de estos atributos y/o métodos en un Main donde se creen 5 aviones y 10 pasajeros.

Debemos mostrar el mensaje que nos indique cuántos objetos de cada clase hemos creado cada vez que creemos un objeto nuevo.

EJERCICIO 3

Crear una clase Vuelo:

Para dicha clase deberemos guardar (al menos):

- El avión con el que se realiza el vuelo.
- La fecha del vuelo.
- El nombre del aeropuerto de origen
- El nombre del aeropuerto de destino
- La lista de pasajeros del vuelo.

Además de los métodos básicos (constructores, getter / setters / equals) que consideréis oportunos debéis implementar el siguiente comportamiento:

- Un método para añadir un pasajero al vuelo. addPassenger(). Se le pasará como parámetro un objeto pasajero. Si el pasajero ya

estaba en la lista de pasajeros deberá mostrar un mensaje de error.
- Un método para borrar un pasajero. delPassenger(). Se le pasará como parámetro el número de pasaporte del pasajero. Si el pasajero no estaba en la lista de pasajeros deberá mostrar un mensaje de error.
- Un método que listPassengers() que muestre por pantalla la lista de pasajeros.

Crear en un Main.java dos vuelos con 5 pasajeros cada uno. Utilizar al menos una vez los métodos solicitados para cada vuelo.

NOTA: Para este ejercicio no se pueden usar collections. Utilizaremos Arrays y los gestionaremos de manera dinámica (el tamaño del array siempre coincidirá con el número de pasajeros),

EJERCICIO 4

Crear una clase Aereopuerto para gestionar los vuelos. De los aeropuertos necesitamos guardar (al menos):

- El nombre del aeropuerto.
- La lista de vuelos del aeropuerto.

Necesitaré además, para definir el comportamiento:

- método para añadir vuelos al aeropuerto (debemos comprobar que el vuelo sale del mismo aeropuerto al que los estamos añadiendo)
- Método para eliminar vuelos de un aeropuerto. Deberá comprobar que el vuelo existe

- Método para mostrar los vuelos de una determinada fecha que pasaré como parámetro. Si no hay vuelos ese día mostrará un mensaje con esa información.

Crear en un Main.java dos vuelos con 2 aeropuertos cada uno con 2 vuelos con 2 pasajeros cada uno. Utilizar al menos una vez los métodos solicitados para aeropuerto.

NOTA: Para este ejercicio no se pueden usar collections. Utilizaremos Arrays y los gestionaremos de manera dinámica (el tamaño del array siempre coincidirá con el número de vuelos),

EJERCICIO 5

La policía durante este confinamiento y para gestionar las posibles multas a aquellos ciudadanos necesita tener a su disposición las salidas de los ciudadanos. Para ello elaboraremos una aplicación.

Desde la aplicación para gestionar todo esto deberemos:

- Insertar los datos de un ciudadano concreto.
- Añadir una salida a un ciudadano para una fecha determinada. Podrá ser (Compra, Deporte, Paseo).
- Comprobar un número de salida que un ciudadano en una fecha determinada.
- Mostrar la multa que se le va que corresponde a un ciudadano. Será 600€ por el número de visitas del mismo tipo (quitando la salida de cada tipo a la que tienen derecho.

NOTA: Tomad las decisiones de diseño que consideréis y justificarlas.

SEMANA 11-17 de MAYO

EJERCICIO 6

Se quiere informatizar una biblioteca. Crea las clases Publicacion, Libro y Revista.

Las clases deben estar implementadas con la jerarquía correcta. Las características comunes de las revistas y de los libros son el código ISBN, el título, y el año de publicación.

Los libros tienen además un atributo prestado. Cuando se crean los libros, no están prestados.

Las revistas tienen un número.

La clase Libro debe implementar la interfaz Prestable que tiene los métodos presta, devuelve y estaPrestado.

Demostrar el uso de toda esta jerarquía en un fichero Main.java

EJERCICIO 7

Añadir una clase que sea padre de todas las anteriores pero que no pueda instanciarse. Dicha clase se llamará ObjetoInventario y tendrá un único atributo fechaCompra.

Demostrar el uso de toda esta jerarquía en un fichero Main.java

EJERCICIO 8

Se trata de realizar una aplicación de gestión en el ámbito inmobiliario. En ella se mantendrán datos sobre viviendas en venta y sobre ofertas que los clientes realicen sobre ellas.

El catálogo de clases de la aplicación será el siguiente:

- Clase “Inmueble”. Atributos: referencia catastral (cadena), año de construcción (entero) y un objeto de la clase “Domicilio”.
- Clase “Domicilio”. Atributos: nombre de vía (cadena), número (entero), otros datos de localización (cadena) y código postal (entero).
- Clase “Vivienda”.
    - Hereda de la clase “Inmueble”.
    - Atributos propios: un objeto de la clase “Persona” que representará el propietario principal, tipo (podrá ser “casa”, “piso” u “otro”) y un marcador de disponibilidad para ser vendida (booleano). Cada vivienda dispondrá también de una lista de ofertas de compra.
    - Implementar el método equals de Object. Dos viviendas serán iguales si tienen la misma referencia catastral.
    - Un método ordenaOfertas(boolean ascendente) que ordene el listado de ofertas por precio ascencente si el parámetro “ascendente” vale true, y descendente si vale “false”. (nota: La ordenación natural de las ofertas es por precio ascendente).
- Clase “Persona”. Atributos: NIF (cadena) y nombre (cadena).
- Clase “Oferta”. Atributos: un objeto de la clase “Persona” que representará la persona que ha efectuado la oferta y un importe (numérico).
- Clase Inmobiliaria. Atributo: lista de viviendas. Métodos:

- Alta de vivienda. Previamente deberá comprobarse si ya existe una vivienda igual, pasándose, en caso de que esté, a “disponible”. Si no está, deberá incluirse en la lista también como disponible.
- Alta de oferta de compra por una vivienda. El método añadirá dicha oferta a la lista de la vivienda correspondiente.
- Baja de una vivienda. Consistirá en hacerla “no disponible”, eliminándose todos los datos sobre ofertas de compra efectuadas por clientes.

- Cree una clase principal donde se prueben todas las funcionalidades implementadas.
- Deberá crear los constructores y métodos que considere necesarios para desarrollar las funcionalidades solicitadas.
- Los métodos requeridos expresamente deberán tener los parámetros que considere necesarios en cada caso, así como devolver lo que se considere oportuno.

EJERCICIO 9

En el mercado disponemos de muchos tipos de mandos a distancia y para montar una tienda de mandos queremos hacer una aplicación para gestionar dicha variedad.

La clase MandoADistancia no será instanciable y tendremos que guardar:

- El modelo del mando
- La anchura del mando
- La altura del mando
- El precio del mando
- Estado del aparato (ENCENDIDO, APAGADO)

Tendrá un método apagar() y otro encender() que establecen correctamente el estado.

La clase MandoTV tendrás dos atributos nuevos: volumen y canal.

La clase MandoMiniCadena tendrá el atributo: volumen.

La clase MandoAspiradora tendrá un atributo nuevo: velocidad.

La clase MandoAireAcondicionado tendrá tres atributos nuevos: modo(FRIO, CALOR), temperatura y velocidad

No todos los mandos realizan las mismas operaciones así que mediante el uso de interfaces debemos:

- Garantizar que podemos subir y bajar el volumen en los mandos correspondientes.
- Garantizar que podemos subir y bajar la velocidad en los mandos correspondientes.

El resto de operaciones no requerirán del uso de interfaces.

Debemos de crear, todos los constructores / getter / setter / equals / toString que consideremos oportunos.

En el programa Main crear un vector de 10 objetos MandosADistancia , rellenadlo y mostrarlo ordenador por precios y por superficie (altura*anchura).

EJERCICIO 10

Nos piden crear una serie de clases para manipular las fichas de los juegos de mesa.

La idea es disponer de una clase abstracta general que englobe el concepto de pieza de tablero y contenga las características básicas inherentes a una pieza de tablero:

posicionX, posicionY, color (puede ser BLANCO o NEGRO).

Sus subclases son:

- Pieza de ajedrez. Nuevamente será una clase abstracta. Se trata de una pieza "móvil", es decir, que se puede mover en el tablero.

- Pieza de damas. Es una clase instanciable, pues pueden existir objetos de tipo "ficha de las damas". También se trata de una pieza que puede ser movida.

- Pieza del juego de los barquitos. También será una clase instanciable, aunque no compartirá con las otras dos anteriores la capacidad de ser móvil, ya que su posición será fija durante toda la partida.

Crea la interfaz Movible, que será implementada únicamente por las clases anteriores que se correspondan con piezas que pueden moverse. Esta interfaz incluirá los siguientes métodos:

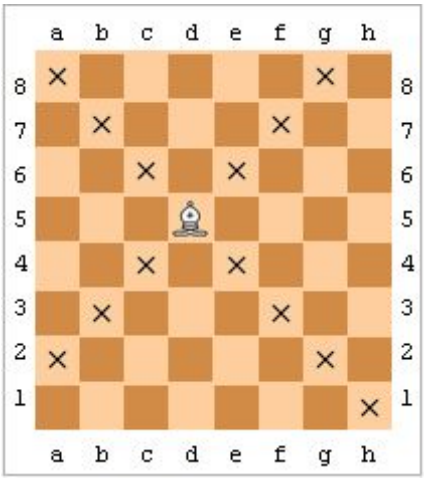
- boolean esMovible (int x, int y). Este método indicará si es posible mover la pieza a las coordenadas que se le indican.

- void mover (int x, int y). Este método moverá (si es posible) la pieza a las coordenadas que se le indican. Si no le es

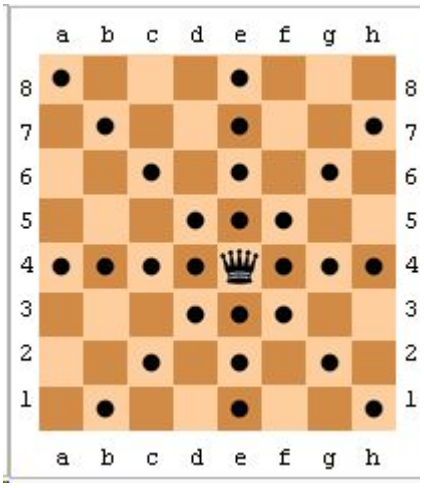
posible, no hará ningún cambio en la pieza.

Ahora bien, cada clase implementará estos métodos de una manera diferente según se trate de un tipo de pieza u otro:

- En el caso de las piezas de las damas, consideraremos válido el movimiento diagonal hacia adelante o hacia atrás tantas posiciones como se quiera:



- En el caso de las piezas del ajedrez, cada pieza (peón, alfil, reina...) tiene unas reglas de movimiento diferentes, por lo que sería necesario definir una clase por cada tipo de pieza diferente del ajedrez para que implemente sus reglas de movimiento. Por razones de tiempo solo implementaremos la clase Reina, cuyo movimiento se realiza de esta forma:



Se indica también que debemos tener una forma de comparar piezas (todas, movibles o no). El orden natural de las piezas viene indicado por su componente Y. Es decir, una pieza se considera menor que otra si su posiciónY es menor que la posiciónY de la otra.

Algunas consideraciones que debemos tener en cuenta sobre el desarrollo de estas clases son:

- Crea los constructores que consideres, teniendo en cuenta que no deben existir constructores que permitan crear piezas con valores inválidos y si ello fuera a suceder, asignaremos valores por defecto a los atributos (los que decidas) .

- Respecto a los métodos get y set de los atributos posicionX, posicionY, color, ¿deberían ser en este caso todos públicos? Por ejemplo, ¿tiene sentido poder hacer un setColor una vez creada la pieza? ¿tiene sentido hacer setPosicionX o para hacer eso deberíamos usar algún método más específico?

- La clase PiezaTablero debe sobrescribir el método toString para que desde el programa principal se pueda mostrar en pantalla el contenido (ubicación y color) de cualquier objeto que herede de ella.

- Para cualquier tipo de pieza (barquitos, ajedrez, damas) se supone un tablero de 8x8 donde la casilla inferior izquierda tendrá las coordenadas (1,1) y la superior derecha (8,8). Esto significa que los métodos de la interfaz Movible deben tenerlo en cuenta a la hora de analizar la posibilidad de un movimiento (para no salirse del tablero).
- Comenta el código con explicaciones útiles sobre las decisiones que hayas tomado, así como el sentido de los atributos, y todo lo que facilite la comprensión del código.

Una vez tengamos lista esta biblioteca de clases tendremos que hacer un pequeño programa que pruebe parte de la funcionalidad de esos elementos. Para ello desarrollaremos una aplicación que rellene un array de piezas movibles con distintos objetos de cada una de las piezas disponibles que pueden moverse para después recorrerlo intentando aplicarles algún movimiento y observar si ese movimiento es posible o no. (Recuerda que una variable de referencia puede ser del tipo de una Interfaz.) Una vez realizados los movimientos posibles, debe mostrarse el array de tal forma que veamos cada una de las piezas.

Crea también otro array de piezas (movibles o no), y ordena las piezas por su orden natural.