EJERCICIOS DE PROGRAMACIÓN ORIENTADA A OBJETOS (hoja II)

EJERCICIO 1. Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular y cantidad (puede tener decimales).

El titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumpla lo anterior.

Crea sus métodos get, set y toString.

Tendrá dos métodos especiales:

ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.

retirar(double cantidad): se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

EJERCICIO 2. Haz una clase llamada Persona que siga las siguientes condiciones:

Sus atributos son: nombre, edad, DNI, sexo (H hombre, M mujer), peso y altura. No queremos que se accedan directamente a ellos. Piensa que modificador de acceso es el más adecuado, también su tipo. Si quieres añadir algún atributo puedes hacerlo.

Por defecto, todos los atributos menos el DNI serán valores por defecto según su tipo (0 números, cadena vacía para String, etc.). Sexo será hombre por defecto, usa una constante para ello.

Se implantaran varios constructores:

Un constructor por defecto.

Un constructor con el nombre, edad y sexo, el resto por defecto.

Un constructor con todos los atributos como parámetro.

Los métodos que se implementaran son:

calcularIMC(): calculara si la persona está en su peso ideal (peso en kg/(altura^2 en m)), si esta fórmula devuelve un valor menor que

20, la función devuelve un -1, si devuelve un número entre 20 y 25 (incluidos), significa que está por debajo de su peso ideal la función devuelve un 0 y si devuelve un valor mayor que 25 significa que tiene sobrepeso, la función devuelve un 1. Te recomiendo que uses constantes para devolver estos valores.

esMayorDeEdad(): indica si es mayor de edad, devuelve un booleano.

comprobarSexo(char sexo): comprueba que el sexo introducido es correcto. Si no es correcto, sera H. No será visible al exterior.

toString(): devuelve toda la información del objeto.

generaDNI(): genera un número aleatorio de 8 cifras, genera a partir de este su número su letra correspondiente. Este método será invocado cuando se construya el objeto. Puedes dividir el método para que te sea más fácil. No será visible al exterior.

Métodos set de cada parámetro, excepto de DNI.

Ahora, crea una clase ejecutable que haga lo siguiente:

Pide por teclado el nombre, la edad, sexo, peso y altura.

Crea 3 objetos de la clase anterior, el primer objeto obtendrá las anteriores variables pedidas por teclado, el segundo objeto obtendrá todos los anteriores menos el peso y la altura y el último por defecto, para este último utiliza los métodos set para darle a los atributos un valor.

Para cada objeto, deberá comprobar si está en su peso ideal, tiene sobrepeso o por debajo de su peso ideal con un mensaje.

Indicar para cada objeto si es mayor de edad.

Por último, mostrar la información de cada objeto.

Puedes usar métodos en la clase ejecutable, para que os sea más fácil.

EJERCICIO 3. Haz una clase llamada Password que siga las siguientes condiciones:

Que tenga los atributos longitud y contraseña. Por defecto, la longitud será de 8.

Los constructores serán los siguientes:

Un constructor por defecto.

Un constructor con la longitud que nosotros le pasemos. Generará una contraseña aleatoria con esa longitud.

Los métodos que implementa serán:

esFuerte(): devuelve un booleano si es fuerte o no, para que sea fuerte debe tener más de 2 mayúsculas, más de 1 minúscula y más de 5 números.

generarPassword(): genera la contraseña del objeto con la longitud que tenga.

Método get para contraseña y longitud.

Método set para longitud.

Ahora, crea una clase clase ejecutable:

Crea un array de Passwords con el tamaño que tu le indiques por teclado.

Crea un bucle que cree un objeto para cada posición del array.

Indica también por teclado la longitud de los Passwords (antes de bucle).

Crea otro array de booleanos donde se almacene si el password del array de Password es o no fuerte (usa el bucle anterior).

Al final, muestra la contraseña y si es o no fuerte (usa el bucle anterior). Usa este simple formato:

contraseña1 valor_booleano1

contraseña2 valor_bololeano2

…

EJERCICIO 4. Crearemos una supeclase
llamada Electrodomestico con las siguientes características:

Sus atributos son precio base, color, consumo energético (letras entre A y F) y peso. Indica que se podrán heredar.

Por defecto, el color sera blanco, el consumo energético será F, el precioBase es de 100 € y el peso de 5 kg. Usa constantes para ello.

Los colores disponibles son blanco, negro, rojo, azul y gris. No importa si el nombre está en mayúsculas o en minúsculas.

Los constructores que se implementaran serán

Un constructor por defecto.

Un constructor con el precio y peso. El resto por defecto.

Un constructor con todos los atributos.

Los métodos que implementara serán:

Métodos get de todos los atributos.

comprobarConsumoEnergetico(char letra): comprueba que la letra es correcta, sino es correcta usara la letra por defecto. Se invocara al crear el objeto y no será visible.

comprobarColor(String color): comprueba que el color es correcto, sino lo es usa el color por defecto. Se invocara al crear el objeto y no será visible.

precioFinal(): según el consumo energético, aumentara su precio, y según su tamaño, también. Esta es la lista de precios:

LETRA	PRECIO
A	100 €
B	80 €
C	60 €
D	50 €
E	30 €
F	10 €

TAMAÑO	PRECIO
Entre 0 y 19 kg	10 €
Entre 20 y 49 kg	50 €
Entre 50 y 79 kg	80 €
Mayor que 80 kg	100 €

Crearemos una subclase llamada Lavadora con las siguientes características:

Su atributo es carga, además de los atributos heredados.

Por defecto, la carga es de 5 kg. Usa una constante para ello.

Los constructores que se implementaran serán:

Un constructor por defecto.

Un constructor con el precio y peso. El resto por defecto.

Un constructor con la carga y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

Método get de carga.

precioFinal():, si tiene una carga mayor de 30 kg, aumentara el precio 50 €, sino es así no se incrementara el precio. Llama al método padre y añade el código necesario. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Crearemos una subclase llamada Television con las siguientes características:

Sus atributos son resolución (en pulgadas) y sintonizador TDT (booleano), ademas de los atributos heredados.

Por defecto, la resolución será de 20 pulgadas y el sintonizador será false.

Los constructores que se implementaran serán:

Un constructor por defecto.

Un constructor con el precio y peso. El resto por defecto.

Un constructor con la resolución, sintonizador TDT y el resto de atributos heredados. Recuerda que debes llamar al constructor de la clase padre.

Los métodos que se implementara serán:

Método get de resolución y sintonizador TDT.

precioFinal(): si tiene una resolución mayor de 40 pulgadas, se incrementara el precio un 30% y si tiene un sintonizador TDT incorporado, aumentara 50 €. Recuerda que las condiciones que hemos visto en la clase Electrodomestico también deben afectar al precio.

Ahora crea una clase ejecutable que realice lo siguiente:

Crea un array de Electrodomesticos de 10 posiciones.

Asigna a cada posición un objeto de las clases anteriores con los valores que desees.

Ahora, recorre este array y ejecuta el método precioFinal().

Deberás mostrar el precio de cada clase, es decir, el precio de todas las televisiones por un lado, el de las lavadoras por otro y la suma de los Electrodomesticos (puedes crear objetos Electrodomestico, pero recuerda que Television y Lavadora también son electrodomésticos). Recuerda el uso operador instanceof.

Por ejemplo, si tenemos un Electrodomestico con un precio final de 300, una lavadora de 200 y una televisión de 500, el resultado final será de 1000 (300+200+500) para electrodomésticos, 200 para lavadora y 500 para televisión.

EJERCICIO 5. Crearemos una clase llamada Serie con las siguientes características:

Sus atributos son título, numero de temporadas, entregado, género y creador.

Por defecto, el número de temporadas es de 3 temporadas y entregado false. El resto de atributos serán valores por defecto según el tipo del atributo.

Los constructores que se implementaran serán:

Un constructor por defecto.

Un constructor con el título y creador. El resto por defecto.

Un constructor con todos los atributos, excepto de entregado.

Los métodos que se implementara serán:

Métodos get de todos los atributos, excepto de entregado.

Métodos set de todos los atributos, excepto de entregado.

Sobrescribe los métodos toString.

Crearemos una clase Videojuego con las siguientes características:

Sus atributos son título, horas estimadas, entregado, género y compañía.

Por defecto, las horas estimadas serán de 10 horas y entregado false. El resto de atributos serán valores por defecto según el tipo del atributo.

Los constructores que se implementaran serán:

Un constructor por defecto.

Un constructor con el título y horas estimadas. El resto por defecto.

Un constructor con todos los atributos, excepto de entregado.

Los métodos que se implementara serán:

Métodos get de todos los atributos, excepto de entregado.

Métodos set de todos los atributos, excepto de entregado.

Sobrescribe los métodos toString.

Como vemos, en principio, las clases anteriores no son padre-hija, pero si tienen en común, por eso vamos a hacer una interfaz llamada Entregable con los siguientes métodos:

entregar(): cambia el atributo prestado a true.

devolver(): cambia el atributo prestado a false.

isEntregado(): devuelve el estado del atributo prestado.

Método compareTo (Object a), compara las horas estimadas en los videojuegos y en las series el número de temporadas. Como parámetro que tenga un objeto, no es necesario que implementes la interfaz Comparable. Recuerda el uso de los casting de objetos.

Implementa los anteriores métodos en las clases Videojuego y Serie. Ahora crea una aplicación ejecutable y realiza lo siguiente:

Crea dos arrays, uno de Series y otro de Videojuegos, de 5 posiciones cada uno.

Crea un objeto en cada posición del array, con los valores que desees, puedes usar distintos constructores.

Entrega algunos Videojuegos y Series con el método entregar().

Cuenta cuantos Series y Videojuegos hay entregados. Al contarlos, devuélvelos.

Por último, indica el Videojuego tiene más horas estimadas y la serie con más temporadas. Muestralos en pantalla con toda su información (usa el método toString()).

EJERCICIO 6. Crear una clase Libro que contenga los siguientes atributos:

– ISBN

– Titulo

– Autor

– Número de páginas

Crear sus respectivos métodos get y set correspondientes para cada atributo. Crear el método toString() para mostrar la información relativa al libro con el siguiente formato:

“El libro con ISBN creado por el autor tiene páginas”

En el fichero main, crear 2 objetos Libro (los valores que se quieran) y mostrarlos por pantalla.

Por último, indicar cuál de los 2 tiene más páginas.

EJERCICIO 7. Vamos a realizar una clase llamada Raices, donde representaremos los valores de una ecuación de 2º grado.
Tendremos los 3 coeficientes como atributos, llamémosles a, b y c.
Hay que insertar estos 3 valores para construir el objeto.
Las operaciones que se podrán hacer son las siguientes:

obtenerRaices(): imprime las 2 posibles soluciones

obtenerRaiz(): imprime única raíz, que será cuando solo tenga una solución posible.

getDiscriminante(): devuelve el valor del discriminante (double), el discriminante tiene la siguiente formula, (b^2)-4*a*c

tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.

tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.

calcular(): mostrara por consola las posibles soluciones que tiene nuestra ecuación, en caso de no existir solución, mostrarlo también.

Formula ecuación 2º grado: (-b±√((b^2)-(4*a*c)))/(2*a)
Solo varia el signo delante de -b

EJERCICIO 8. Queremos representar con programación orientada a objetos, un aula con estudiantes y un profesor.
Tanto de los estudiantes como de los profesores necesitamos saber su nombre, edad y sexo. De los estudiantes, queremos saber también su calificación actual (entre 0 y 10) y del profesor que materia da.
Las materias disponibles son matemáticas, filosofía y física.
Los estudiantes tendrán un 50% de hacer novillos, por lo que si

hacen novillos no van a clase pero aunque no vayan quedara registrado en el aula (como que cada uno tiene su sitio).
El profesor tiene un 20% de no encontrarse disponible (reuniones, baja, etc.)
Las dos operaciones anteriores deben llamarse igual en Estudiante y Profesor (polimorfismo).
El aula debe tener un identificador numérico, el número máximo de estudiantes y para que esta destinada (matemáticas, filosofía o física). Piensa que más atributos necesita.
Un aula para que se pueda dar clase necesita que el profesor esté disponible, que el profesor de la materia correspondiente en el aula correspondiente (un profesor de filosofía no puede dar en un aula de matemáticas) y que haya más del 50% de alumnos.
El objetivo es crear un aula de alumnos y un profesor y determinar si puede darse clase, teniendo en cuenta las condiciones antes dichas.
Si se puede dar clase mostrar cuantos alumnos y alumnas (por separado) están aprobados de momento (imaginad que les están entregando las notas).
NOTA: Los datos pueden ser aleatorios (nombres, edad, calificaciones, etc.) siempre y cuando tengan sentido (edad no puede ser 80 en un estudiante o calificación ser 12).

EJERCICIO 9. Nos piden hacer un programa orientado a objetos sobre un cine (solo de una sala) tiene un conjunto de asientos (8 filas por 9 columnas, por ejemplo).
Del cine nos interesa conocer la película que se está reproduciendo y el precio de la entrada en el cine.

De las películas nos interesa saber el título, duración, edad mínima y director.

Del espectador, nos interesa saber su nombre, edad y el dinero que tiene.

Los asientos son etiquetados por una letra (columna) y un número (fila), la fila 1 empieza al final de la matriz como se muestra en la tabla. También deberemos saber si está ocupado o no el asiento.

8 A 8 B 8 C 8 D 8 E 8 F 8 G 8 H 8 I
7 A 7 B 7 C 7 D 7 E 7 F 7 G 7 H 7 I
6 A 6 B 6 C 6 D 6 E 6 F 6 G 6 H 6 I

5 A 5 B 5 C 5 D 5 E 5 F 5 G 5 H 5 I
4 A 4 B 4 C 4 D 4 E 4 F 4 G 4 H 4 I
3 A 3 B 3 C 3 D 3 E 3 F 3 G 3 H 3 I
2 A 2 B 2 C 2 D 2 E 2 F 2 G 2 H 2 I
1 A 1 B 1 C 1 D 1 E 1 F 1 G 1 H 1 I

Realizaremos una pequeña simulación, en el que generaremos muchos espectadores y los sentaremos aleatoriamente (no podemos donde ya este ocupado).
En esta versión sentaremos a los espectadores de uno en uno.

Solo se podrá sentar si tienen el suficiente dinero, hay espacio libre y tiene edad para ver la película, en caso de que el asiento este ocupado le buscamos uno libre.
Los datos del espectador y la película pueden ser totalmente aleatorios.

EJERCICIO 10. Vamos a hacer una baraja de cartas españolas orientado a objetos.

Una carta tiene un número entre 1 y 12 (el 8 y el 9 no los incluimos) y un palo (espadas, bastos, oros y copas).

La baraja estará compuesta por un conjunto de cartas, 40 exactamente.

Las operaciones que podrá realizar la baraja son:

barajar: cambia de posición todas las cartas aleatoriamente.

siguienteCarta: devuelve la siguiente carta que está en la baraja, cuando no haya más o se haya llegado al final, se indica al usuario que no hay más cartas.

cartasDisponibles: indica el número de cartas que aún puede repartir.

darCartas: dado un número de cartas que nos pidan, le devolveremos ese número de cartas (piensa que puedes devolver). En caso de que haya menos cartas que las pedidas, no devolveremos nada pero debemos indicárselo al usuario.

cartasMonton: mostramos aquellas cartas que ya han salido, si no ha salido ninguna indicárselo al usuario

mostrarBaraja: muestra todas las cartas hasta el final. Es decir, si se saca una carta y luego se llama al método, este no mostrara esa primera carta.

EJERCICIO 11. Estando en un grupo de amigos, se planea hacer una porra de la liga de fútbol. A nosotros se nos ocurre hacer un programa en POO para simular como podría desarrollarse la porra.

Cada jugador de la porra, pone un 1 euro cada jornada y decide el resultado de los partidos acordados.

Si nadie acierta en una jornada los resultados, el bote se acumula.

En principio, deben acertar el resultado de dos partidos para llevarse el dinero del bote de la porra.

Todos empiezan con un dinero inicial que será decidido por el programador (ya sea como parámetro o como constante). Si el jugador no tiene dinero en una jornada no podrá jugar la porra.

Para esta versión, entre jugadores podrán repetir resultados repetidos, por lo que el jugador que primero diga ese resultado (tal como estén de orden) se llevara primero el bote.

Los resultados de la porra serán generados aleatoriamente, tanto los de jugador como los de los partidos (no es necesario nombre, solo resultados).

Al final del programa, se deberá mostrar el dinero que tienen los jugadores y el número de veces que han ganado.

Para este ejercicio, recomiendo usar interfaces (no hablo de interfaces gráficas) para las constantes y métodos que sean necesarios.

EJERCICIO 12. Vamos a hacer el juego de la ruleta rusa en Java.

Como muchos sabéis, se trata de un número de jugadores que con un revolver con un sola bala en el tambor se dispara en la cabeza.

Las clases a hacer son:

Revolver:

Atributos:

posición actual (posición del tambor donde se dispara, puede que esté la bala o no)

posición bala (la posición del tambor donde se encuentra la bala)

Estas dos posiciones, se generaran aleatoriamente.

Funciones:

disparar(): devuelve true si la bala coincide con la posición actual

siguienteBala(): cambia a la siguiente posición del tambor

toString(): muestra información del revolver (posición actual y donde está la bala)

Jugador:

Atributos

id (representa el número del jugador, empieza en 1)

nombre (Empezara con Jugador más su ID, “Jugador 1” por ejemplo)

vivo (indica si está vivo o no el jugador)

Funciones:

disparar(Revolver r): el jugador se apunta y se dispara, si la bala se dispara, el jugador muere.

Juego:

Atributos:

Jugadores (conjunto de Jugadores)

Revolver

Funciones:

finJuego(): cuando un jugador muere, devuelve true

ronda(): cada jugador se apunta y se dispara, se informara del estado de la partida (El jugador se dispara, no ha muerto en esa ronda, etc.)

El número de jugadores será decidido por el usuario, pero debe ser entre 1 y 6. Si no está en este rango, por defecto será 6.

En cada turno uno de los jugadores, dispara el revólver, si este tiene la bala el jugador muere y el juego termina.

Aunque no lo haya comentado, recuerda usar una clase ejecutable para probarlo.

EJERCICIO 13. Nos piden hacer una un programa que gestione empleados.

Los empleados se definen por tener:

Nombre

Edad

Salario

También tendremos una constante llamada PLUS, que tendrá un valor de 300€

Tenemos dos tipos de empleados: repartidor y comercial.

El comercial, aparte de los atributos anteriores, tiene uno más llamado comisión (double).

El repartidor, aparte de los atributos de empleado, tiene otro llamado zona (String).

Crea sus constructores, getters and setters y toString (piensa como aprovechar la herencia).

No se podrán crear objetos del tipo Empleado (la clase padre) pero si de sus hijas.

Las clases tendrán un método llamado plus, que según en cada clase tendrá una implementación distinta. Este plus básicamente aumenta el salario del empleado.

En comercial, si tiene más de 30 años y cobra una comisión de más de 200 euros, se le aplicara el plus.

En repartidor, si tiene menos de 25 y reparte en la “zona 3”, este recibirá el plus.

Puedes hacer que devuelva un booleano o que no devuelva nada, lo dejo a tu elección.

Crea una clase ejecutable donde crees distintos empleados y le apliques el plus para comprobar que funciona.

EJERCICIO 14. Nos piden hacer que gestionemos una serie de productos.

Los productos tienen los siguientes atributos:

Nombre

Precio

Tenemos dos tipos de productos:

Perecedero: tiene un atributo llamado días a caducar

No perecedero: tiene un atributo llamado tipo

Crea sus constructores, getters, setters y toString.

Tendremos una función llamada calcular, que según cada clase hará una cosa u otra, a esta función le pasaremos un numero siendo la cantidad de productos

En Producto, simplemente seria multiplicar el precio por la cantidad de productos pasados.

En Perecedero, aparte de lo que hace producto, el precio se reducirá según los días a caducar:

Si le queda 1 día para caducar, se reducirá 4 veces el precio final.

Si le quedan 2 días para caducar, se reducirá 3 veces el precio final.

Si le quedan 3 días para caducar, se reducirá a la mitad de su precio final.

En NoPerecedero, hace lo mismo que en producto

Crea una clase ejecutable y crea un array de productos y muestra el precio total de vender 5 productos de cada uno. Crea tú mismo los elementos del array.

EJERCICIO 15. Nos piden hacer un almacén, vamos a usar programación orientado a objetos.

En un almacén se guardan un conjunto de bebidas.

Estos productos son bebidas como agua mineral y bebidas azucaradas (coca-cola, fanta, etc). De los productos nos interesa saber su identificador (cada uno tiene uno distinto), cantidad de litros, precio y marca.

Si es agua mineral nos interesa saber también el origen (manantial tal sitio o donde sea).

Si es una bebida azucarada queremos saber el porcentaje que tiene de azúcar y si tiene o no alguna promoción (si la tiene tendrá un descuento del 10% en el precio).

En el almacén iremos almacenado estas bebidas por estanterías (que son las columnas de la matriz).

Las operaciones del almacén son las siguientes:

Calcular precio de todas las bebidas: calcula el precio total de todos los productos del almacén.

Calcular el precio total de una marca de bebida: dada una marca, calcular el precio total de esas bebidas.

Calcular el precio total de una estantería: dada una estantería (columna) calcular el precio total de esas bebidas.

Agregar producto: agrega un producto en la primera posición libre, si el identificador esta repetido en alguno de las bebidas, no se agregará esa bebida.

Eliminar un producto: dado un ID, eliminar el producto del almacén.

Mostrar información: mostramos para cada bebida toda su información.

Puedes usar un main para probar las funcionalidades (añade productos, calcula precios, muestra información, etc)

EJERCICIO 16. Nos piden realizar una agenda telefónica de contactos.

Un contacto está definido por un nombre y un teléfono (No es necesario de validar). Un contacto es igual a otro cuando sus nombres son iguales.

Una agenda de contactos está formada por un conjunto de contactos (Piensa en que tipo puede ser)

Se podrá crear de dos formas, indicándoles nosotros el tamaño o con un tamaño por defecto (10)

Los métodos de la agenda serán los siguientes:

aniadirContacto(Contacto c): Añade un contacto a la agenda, sino se pueden meter más a la agenda se indicara por pantalla. No se pueden meter contactos que existan, es decir, no podemos duplicar nombres, aunque tengan distinto teléfono.

existeContacto(Conctacto c): indica si el contacto pasado existe o no.

listarContactos(): Lista toda la agenda

buscaContacto(String nombre): busca un contacto por su nombre y muestra su teléfono.

eliminarContacto(Contacto c): elimina el contacto de la agenda, indica si se ha eliminado o no por pantalla

agendaLlena(): indica si la agenda está llena.

huecosLibres(): indica cuantos contactos más podemos meter.

Crea un menú con opciones por consola para probar todas estas funcionalidades.

EJERCICIO 17. Vamos a hacer unas mejoras a la clase Baraja del ejercicio 5 de POO de los videos.

Lo primero que haremos es que nuestra clase Baraja será la clase padre y será abstracta.

Le añadiremos el número de cartas en total y el número de cartas por palo.

El método crearBaraja() será abstracto.

La clase Carta tendrá un atributo genérico que será el palo de nuestra versión anterior.

Creamos dos Enum:

PalosBarEspañola:

OROS

COPAS

ESPADAS

BASTOS

PalosBarFrancesa:

DIAMANTES

PICAS

CORAZONES

TREBOLES

Creamos dos clases hijas:

BarajaEspañola: tendrá un atributo boolean para indicar si queremos jugar con las cartas 8 y 9 (total 48 cartas) o no (total 40 cartas).

BarajaFrancesa: no tendrá atributos, el total de cartas es 52 y el número de cartas por palo es de 13. Tendrá dos métodos llamados:

cartaRoja(Carta<PalosBarFrancesa> c): si el palo es de corazones y diamantes.

cartaNegra(Carta<PalosBarFrancesa> c): si el palo es de tréboles y picas.

De la carta modificaremos el método toString()

Si el palo es de tipo PalosBarFrancesa:

La carta número 11 será Jota

La carta numero 12 será Reina

La carta numero 13 será Rey

La carta numero 1 será As

Si el palo es de tipo PalosBarFrancesa:

La carta numero 10 será Sota

La carta numero 12 será Caballo

La carta numero 13 será Rey

La carta numero 1 será As

EJERCICIO 18. Una academia nos pide hacer un programa para hacer un pequeño test a sus alumnos.

Estas preguntas, para facilitar la inclusión, estarán escritas en un txt (incluido en la descarga del proyecto).

Una opción se compone de:

El texto de la opción (digamos la respuesta)

Es correcto o no

Una pregunta consta de:

Pregunta (tendrá delante dos puntos y coma ;P;)

Opciones de la pregunta (entre 2 y 4)

Opción correcta (tendrá delante dos puntos y coma ;R;)

Puntos

La pregunta no será válida en los siguientes casos:

Las opciones no están entre 2 y 4.

La opción correcta esta entre el número de opciones y es un número.

Los puntos es un número entero.

Sus métodos son:

mostrarPregunta(): muestra la pregunta con sus opciones.

comprobarRespuesta(int respuestaUsuario): comprueba la respuesta del usuario si es correcta o no.

Getter de los atributos.

Un test está formado por un conjunto preguntas y los puntos acumulados. Piensa que debemos saber por cual pregunta vamos.

Sus métodos son:

cargarPreguntas(String fichero): carga todas las preguntas del fichero

siguientePregunta(): devuelve la siguiente pregunta

reiniciarTest(): nos permite reiniciar el test.

realizarTest(): empieza el test y empieza a formular las preguntas

El fichero de preguntas tiene el siguiente formato:

;P;Pregunta 1

Opción 1 pregunta 1

Opción 2 pregunta 1

Opción 3 pregunta 1

Opción 4 pregunta 1

;R;Numero opción correcta

Puntos pregunta 1

;P;Pregunta 2

Opción 1 pregunta 2

Opción 2 pregunta 2

…

…