Ejercicios Java

Clases y Objetos

Ejercicio 1: Animal

Definir una clase llamada `Animal` según las especificaciones siguientes:

- **Atributos de instancia:**
  - `nombre`, de tipo `String`
  - `fecha`, de tipo `LocalDate`
- **Métodos de instancia:**
  - Constructor que inicialice el estado del objeto con los valores que reciba a través de sus parámetros formales.
  - Constructor que inicialice el estado del objeto con un nombre que reciba a través de un parámetro formal y con la fecha actual.
  - Métodos `get` y `set`.
  - Redefinir el método toString para que retorne la cadena:

    `"Nombre: nombre - Edad: edad"`

Escribir un programa que ponga a prueba la clase `Animal` utilizando todos sus constructores y métodos.

Ejercicio 2: Hucha

Definir una clase llamada `Hucha` según las especificaciones siguientes:

- La hucha solo admite monedas de uno y dos euros y billetes de 5, 10, 20 y 50 euros.
- La hucha podrá estar abierta o cerrada. Para abrir o cerrar la hucha se requiere una contraseña.
- La contraseña se podrá cambiar en cualquier momento.
- La hucha podrá proporcionar información acerca de la cantidad de monedas o billetes que almacena de cada clase, así como el valor total en euros.
- Se podrán retirar de la hucha monedas o billetes indicando el tipo y la cantidad de los mismos que se desea retirar. Si la cantidad solicitada supera la cantidad almacenada se retirará únicamente la cantidad disponible.
- Se podrá retirar de la hucha una cantidad de euros específica minimizando el número de monedas y billetes que se retirarán.
- Se podrá construir una hucha vacía, una hucha con una cantidad específica de monedas y billetes de cada tipo o una hucha con una cantidad inicial de euros que se desglosará utilizando la mínima cantidad de monedas y billetes.

Escribir un programa que ponga a prueba la clase `Hucha` utilizando todos sus constructores y métodos.

Ejercicio 3: Autor

Definir una clase llamada `Autor` según las especificaciones siguientes:

- **Atributos de instancia:**
  - `nombre`, de tipo `String`
  - `email`, de tipo `String`
  - `género` (para almacenar el género: masculino o femenino), de tipo String
- **Métodos de instancia:**
  - constructor que inicialice el estado del objeto con los valores que reciba a través de sus parámetros formales.
  - `Getters` para obtener el nombre, el email y el género de una instancia de la clase.

- `Setter` para cambiar el correo electrónico de las instancias de la clase.
- Redefinir el método `toString` para que retorne una cadena que contenga una breve descripción del estado del objeto que lo invoca con el formato: "*`nombre_autor (género) email`*". Ejemplo:

  `"Alberto Fernández (masculino) afernandez@mail.com"`

Escribir un programa que ponga a prueba la clase `Autor` utilizando todos sus constructores y métodos.

Ejercicio 4: Libro

Definir una clase llamada `Libro` según las especificaciones siguientes:

- **Atributos de instancia:**
  - `título`, de tipo `String`
  - `autores`, de tipo `Collection<Autor>`
  - `precio`, de tipo `float`
  - `stock`, de tipo `int`
- **Métodos de instancia:**
  - Constructor que inicialice el estado del objeto con los datos recibidos en los parámetros formales siguientes: `título` de tipo `String`, autores de tipo `Collection<Autor>` y precio de tipo `float`. El atributo `stock` se inicializará con el valor por defecto.
  - Constructor que inicialice todos los atributos con datos recibidos a través de parámetros formales.
  - Getters
  - Setters para los atributos `precio` y `stock`.
  - Redefinir el método toString para que retorne información con el formato:

    "*título_libro* (autor1, autor2, …) *precio* € - *stock* unidades en stock"

Escribir un programa que ponga a prueba la clase `Libro` utilizando todos sus constructores y métodos.

Ejercicio 5: Naipe

Definir la clase `Naipe` para poder representar naipes de la baraja inglesa según las especificaciones siguientes:

- Cada naipe pertenece a un palo: tréboles ♣, diamantes ♦, corazones ♥ o picas ♠.
- Cada naipe tiene un rango que se corresponde con un número del 2 al 10 o con una de las figuras Ace (A), Jack (J), Queen (Q) o King (K).
- Cada naipe podrá tener un valor numérico asociado a cada rango que podrá variar en función de los diferentes juegos de naipes.

Ejercicio 6: Mazo

Definir la clase `Mazo` para poder representar un mazo de naipes según las especificaciones siguientes:

- Un objeto Mazo almacenará una colección de naipes (usar un ArrayList).
- **Constructores:**
  - Constructor sin parámetros que crea un mazo en el que se encuentran los 13 naipes de cada uno de los cuatro palos (una baraja de 52 naipes).

    - Constructor que recibe un parámetro `n` de tipo `int` para construir un mazo formado por `n` barajas.
- **Métodos de instancia:**
    - get: retorna un naipe elegido de forma aleatoria sin retirarlo del mazo.
    - remove: retorna un naipe elegido de forma aleatoria retirándolo del mazo.
    - add: añade un naipe al mazo.
    - addAll: añade al mazo los naipes almacenados en otro mazo que se recibe a través de un parámetro formal. Una vez añadidos se eliminan del mazo original.
    - Añadir cualesquiera métodos adicionales que se consideren útiles.

Escribir un programa que utilizando las clases `Naipe` y `Mazo` permita jugar a la carta más alta con el ordenador como oponente, teniendo en cuenta que los naipes se ordenan de menor a mayor valor de la forma siguiente: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A

Ejercicio 7: Automóvil

Definir una clase **Automovil** según las especificaciones siguientes:

- Cada objeto **Automovil** permitirá llevar el control del combustible que consume en los trayectos que realiza.
- Los atributos con los que se describe el estado de cada objeto **Automovil** son:
    - Modelo.
    - Capacidad del depósito de combustible en litros.
    - Cantidad de combustible en el depósito en litros.
    - Consumo por kilómetro en litros.
    - Cantidad total de kilómetros recorridos.
    - Cantidad total de combustible consumido.
- Constructores:
    - Uno que inicialice mediante sus parámetros los atributos modelo, capacidad del depósito, cantidad inicial de litros en el depósito y consumo. El resto de atributos tendrán el valor 0.
    - Uno que inicialice mediante sus parámetros los atributos modelo, capacidad del depósito y consumo. La cantidad de combustible en el depósito se inicializará con el valor máximo y el resto de atributos con el valor 0.
- Método **llenarDeposito**. Este método estará sobrecargado:
    - Una versión que llene el depósito hasta el límite.
    - Otra que llene el depósito con una cantidad de litros específica. Si se excede la capacidad del depósito, se llenará el depósito retornando la cantidad de combustible sobrante.
- Método **desplazar** que reciba la cantidad de kilómetros que ha de recorrer el automóvil. Este método comprobará si es posible el desplazamiento con el combustible que hay en el depósito. Si es posible, descontará del depósito el combustible que corresponda y retornará dicha cantidad. Si no es posible tendrá que ponerlo de manifiesto de alguna forma.

- Métodos **get** y **set** que estimes oportuno.

NO SE PERMITEN EN NINGÚN MÉTODO DE LA CLASE SENTENCIAS QUE ENVÍE DATOS A LA SALIDA ESTÁNDAR.

Desarrollar un programa que ponga a prueba la clase **Automovil** con datos que se recibirán a través de la entrada estándar según las especificaciones siguientes:

- Número ***N*** de automóviles en la primera línea.
- En las ***N*** líneas siguientes se recibirán los datos de los automóviles a razón de uno por línea con el formato siguiente: ***<modelo> <capacidad del depósito> <litros en el depósito> <consumo>***. No se repetirá ningún modelo, es decir, solo se creará un automóvil de cada modelo. Se almacenarán en un array todos los vehículos creados.
- En el resto de las líneas excepto la última, se recibirán datos de los desplazamientos que han de realizar los automóviles con el formato siguiente: **desplazar *<modelo> <kilómetros>***. Se pueden especificar varios desplazamientos para el mismo modelo en diferentes líneas. Por cada línea leída se ejecutará el desplazamiento inmediatamente y se mostrará en la salida estándar el resultado de la forma siguiente:
  - Si el desplazamiento ha sido posible, se mostrará el modelo, el combustible que queda en el depósito y el combustible consumido.
  - Si el desplazamiento no es posible, se mostrará el mensaje “Combustible insuficiente para este desplazamiento”.
- En la última línea, la palabra “fin”.

Después de procesar todas las líneas de entrada, Se mostrará por cada vehículo el modelo, el combustible que le queda, la cantidad total de kilómetros recorridos y el total de litros de combustible consumidos.

Ejemplos:

Entrada	Salida
2 AudiA4 80 23 0.3 BMW-M2 75 45 0.42 desplazar AudiA4 5 desplazar BMW-M2 56 desplazar AudiA4 13 fin	AudiA4 21.50 1.50 BMW-M2 21.48 23.52 AudiA4 17.60 3.90 ------------------ AudiA4 17.60 18 5.40 BMW-M2 21.48 56 23.52
3 AudiA4 80 18 0.34 BMW-M2 75 33 0.41 Ferrari-488Spider 70 50 0.47 desplazar Ferrari-488Spider 97 desplazar AudiA4 85 desplazar Ferrari-488Spider 35 desplazar AudiA4 50 fin	Ferrari-488Spider 4.41 45.59 Combustible insuficiente para este desplazamiento Combustible insuficiente para este desplazamiento AudiA4 1.00 17.00 ------------------------------ AudiA4 1.00 50 17.00 BMW-M2 33.00 0 0.00 Ferrari-488Spider 4.41 97 45.59

Ejercicio 8: Pokemon

Se desea desarrollar un programa que simule torneos de pokemons. Para comenzar, habra que desarrollar dos clases, además de la clase principal:

- **Entrenador:** cada entrenador tiene un nombre, un número de insignias y una colección de pokemons (usar un ArrayList).
- **Pokemon**: cada pokemon tiene un nombre, está ligado a un elemento fundamental y tiene un determinado número de puntos de salud.

Para simular los torneos, el programa recibe a través de la entrada estándar una serie de datos que tendrá que procesar para obtener el resultado de cada torneo. Las especificaciones de entrada son las siguientes:

- Cada entrenador tiene un nombre único y comienza con cero insignias.
- Se recibirán un número indeterminado de líneas en la entrada estándar seguidas de una linea con la palabra "torneo". Cada una de estas líneas contiene información acerca de un pokemos y del entrenador que lo ha capturado con el formato: ***<nombre del entrenador> <nombre del pokemon> <elemento fundamental del pokemon> <salud del pokemon>***.
- A la línea con la plabra "torneo" le sigue un número indeterminado de líneas, cada una conteniendo el nombre de uno de los elementos fundamentales (fuego, agua, electricidad, …) seguidas de una línea con la palabra "fin". Por cada una de esta líneas se ha de comprobar, por cada entrenador, si éste tiene al menos un pokemon ligado al elemento leido. Si es así, recibirá una insignia. En caso contrario, todos sus pokemons perderán 10 puntos de salud y aquellos que llegen a cero puntos de salud morirán, por lo que tendrán que ser eliminados de la colección del entrenador.
- Cuando se lee la línea con la palabra "fin", el torneo finaliza y se mostrará una lista de entrenadores en la que en cada línea se visualizara el nombre de un entrenador, su cantidad de insignias y el número de pokemos que le quedan.
- **OPCIONAL (esto se ve en la 3ª evaluación):** mostrar la lista de entrenadores ordenada por número de insignias, y en caso de empate, por número de pokemos restantes.

Ejemplos:

Input	Output
Ash Charizard fuego 100 Brock Squirtle agua 38 Ash Pikachu electricidad 10 torneo fuego electricidad fin	Ash 2 2 Brock 0 1
Misty Blastoise agua 18 Clemont Pikachu electricidad 22 Millo Kadabra psíquico 90 torneo fuego electricidad fuego End	Clemont 1 1 Misty 0 0 Millo 0 1