EJERCICIO 1

Crea un método llamado **getMayusculas** que reciba una cadena por parámetro y devuelva otra cadena que será el resultado de quitar los espacios en blanco de los extremos de la primera y de pasarla luego todo a mayúsculas.

Pruébalo desde el método main con algún test.

EJERCICIO 2

Crea un método llamado **getMinusculas** que reciba una cadena por parámetro y devuelva otra cadena que será el resultado de quitar los espacios en blanco de los extremos de la primera y de pasarla luego todo a minúsculas.

Pruébalo desde el método main con algún test.

EJERCICIO 3

Crea un método llamado **getMayMin** que reciba dos parámetros:

- Un número que podrá ser 1 o 2
- Una cadena

Tendrá que coger la cadena recibida y quitarle los espacios en blanco del principio y del final. Luego, si el número recibido es 1, devolverá esa cadena en mayúsculas. Si el número recibido es 2, devolverá esa cadena en minúsculas.

Pruébalo desde el método main con algún test.

EJERCICIO 4

Repite el ejercicio anterior, pero reutilizando (llamando a) los métodos creados en el Ejercicio 1 y 2.

EJERCICIO 5

Crea un método llamado **sumar** que reciba dos números enteros y devuelva el resultado de la suma. Pruébalo desde el método main con algún test.

EJERCICIO 6

Lleva el método realizado en el ejercicio anterior a una nueva clase llamada "**Calculadora**". Prueba su funcionamiento desde el método main con algún test.

EJERCICIO 7

Crea un nuevo método en la clase "**Calculadora**" que sea **restar**, otro **multiplicar** y otro **dividir**. Todos reciben por parámetro dos enteros y deben devolver un entero con el resultado de la operación.

EJERCICIO 8

Crea un nuevo método en la clase "**Calculadora**" que sea **calcular**. Recibirá tres parámetros:

- Una cadena que será "ADD", "SUB", "MUL" o "DIV"
- Dos números enteros

En función de la cadena recibida, se tendrá que sumar, restar, multiplicar o dividir los números recibidos. Reutiliza los métodos ya creados en los ejercicios anteriores.

Declara las cadenas "ADD", "SUB", "MUL" o "DIV" como constantes.

Prueba el método nuevo con varias pruebas desde el main (usa las constantes).

EJERCICIO 9

Crea un método llamado **imprimirArray** en una clase llamada **ArrayUtils** que reciba un array de cadenas y lo imprima por completo.

Pruébalo desde el método main con algún test.

EJERCICIO 10

En la clase **ArrayUtils**, crea un método llamado **obtenerPalabra** que reciba un array de cadenas y un número N por parámetros. Debe devolver la cadena que está en la posición N del array. Si N no es una posición válida del array, tendrá que devolver una cadena vacía.

Pruébalo desde el método main con algún test.

EJERCICIO 11

En la clase **ArrayUtils**, crea un método llamado **buscarPalabra** que reciba un array de cadenas y una palabra por parámetros. Debe devolver un entero indicando la posición donde se encuentra esa palabra dentro del array. Si no existe la palabra en el array, devolverá -1.

Pruébalo desde el método main con algún test.

EJERCICIO 12

Crea una clase llamada **Numeros** y dentro crea un método llamado **sumarNumero** que permita sumar todos los números de 1 a N de manera recursiva. Pruébalo desde el main con algún test.

EJERCICIO 13

En la misma clase que el ejercicio anterior, crea un método llamado **factorial** que calcule el factorial de un número N de manera recursiva. Pruébalo desde el main con algún test.

EJERCICIO 14

En la clase **ArrayUtils**, crea un método que imprima un array de cadenas de manera recursiva. Pruébalo desde el main con algún test.

EJERCICIO 15

En la clase **ArrayUtils**, crea un método que sume todos los elementos de un array de enteros de manera recursiva. Pruébalo desde el main con algún test.

EJERCICIO 16

En la clase **ArrayUtils**, crea un método que busque una palabra en un array de cadenas de manera recursiva. Tendrá que devolver la posición donde se encuentra. Si la palabra no existe (no se encuentra) devolverá -1. Pruébalo desde el main con algún test.

EJERCICIO 17

En la clase **ArrayUtils**, crea un método que calcule el máximo número de un array de enteros de manera recursiva. Tendrá que devolver ese máximo. Pruébalo desde el main con algún test.

EJERCICIO 18

Crea una clase llamada Alumno que tenga como atributos privados el dni, nombre, edad y nota. Además, tendrá métodos get y set para todos los atributos. Luego crea un programa que lea por consola los datos de un alumno y los registre en un objeto de la clase que acabas de crear.

EJERCICIO 19

Añade a la clase del ejercicio anterior un método público aprobar() que establecerá la nota en un 5. Úsalo desde el programa que has creado para probar.

EJERCICIO 20

Cambia el constructor de la clase para que reciba un parámetro String que sea el dni del alumno. Es decir, cuando queramos crear un objeto Alumno, habrá que indicar obligatoriamente su DNI. Modifica el programa que has creado para probar para que funcione con este nuevo constructor.

EJERCICIO 21

Crea una clase Persona que tenga los atributos nombre y edad privados con sus get y set. Haz que la clase Alumno herede de ella. Borra todo lo que sea duplicado. ¿Tienes que cambiar algo en tu programa de pruebas?

EJERCICIO 22

Crea una clase Curso que tenga los atributos privados identificador y descripción. Añade los métodos get y set. Haz que la clase Alumno tenga un atributo de tipo Curso. Recuerda crear su get/set

Adapta el programa que has creado para probar para que también solicite el curso.

EJERCICIO 23

Crea una clase Profesor vacía que herede de Persona. Crea un programa que solicite por pantalla todos los datos de un profesor nuevo. El programa tendrá que crear un objeto Profesor y meter todos esos datos que el usuario le proporciona.

EJERCICIO 24

Crea métodos toString() para las clases que has creado. Úsalos en tus programas para imprimir al final el objeto que has creado (tanto el profesor como el alumno)

EJERCICIO 25

Crea método equals() en la clase Alumno para definir que un alumno será igual a otro si tienen el mismo DNI.

Desde tu programa, crea un array de 3 Alumnos. Todos los alumnos pertenecen al curso con identificador = 1 y descripción “DAM-DAW”. El resto de datos de cada alumno, debes solicitarlos al usuario.

Cuando tengas el array completo, comprueba si hay alumnos repetidos dentro del array (ten en cuenta que dos alumnos estarán repetidos en función de lo que se haya implementado en el método equals). Si hay alumnos repetidos, muestra un mensaje de error al usuario.

EJERCICIO 26

Modifica el método setDni() y el constructor de la clase Alumno() para que cada vez que se establezca un DNI al alumno este se registre siempre en mayúsculas.

EJERCICIO 27

Crea un método validarDNI() en la clase Alumno que devuelva un Boolean indicando si el dni que tiene establecido el alumno es correcto o no. Para ello, tendrás que validar lo siguiente:

a) El dni no puede ser null ni vacío
b) El dni tiene que tener una longitud total de 9.

Modifica el programa del ejercicio 25 para validar el DNI de los alumnos que vas creando. Si alguno no es correcto, vuelve a solicitarlo.

EJERCICIO 28

Buscando en internet, nos encontramos este trozo de código que asegura validar un DNI que no sea null:

```
String ejemploDni = "00000000T";
Pattern patron = Pattern.compile("[0-9]{7,8}[A-Z a-z]");
Matcher match = patron.matcher(ejemploDni);
if(match.matches()) {
	System.out.println("El dni es correcto");
}
else {
	System.out.println("El dni es incorrecto");
}
```

Cambia tu método validarDNI() para que realice la segunda parte de la validación (b) utilizando el código que hemos encontrado. Tendrás que adaptarlo, ya que nosotros no queremos imprimir nada. El método tiene que seguir devolviendo un Boolean.

EJERCICIO 29

Crea un método validar() en la clase Alumno que devuelva un Boolean indicando si todos los datos del alumno son válidos o no. Para que los datos de un alumno sean correctos se deben dar estas circunstancias:

- El DNI debe ser válido de acuerdo con el método validarDni() que ya tenemos.
- El curso no puede ser null
- El nombre no puede ser null y tiene que tener una longitud mínima de 10 caracteres.
- La edad no puede ser null y tendrá que estar comprendida entre 12 y 65 (ambas inclusive).

Modifica el programa del ejercicio 25 para que al final del todo recorra el array de alumnos y compruebe uno a uno si son válidos. Si alguno no lo es, avisa al usuario por consola.

EJERCICIO 30

Añade a la clase Curso un atributo llamado "alumnos" que sea un array de Alumno. Debe ser privado. En el constructor de la clase Curso, haz que se reciba el número de alumnos totales que va a tener, de ese modo podrás inicializar el array de alumnos en el Constructor.

Añade un método get() para el nuevo atributo, pero no un set().

Añade además otro método que sea addAlumno() que recibe por parámetro un alumno. El método debe añadir el alumno al array en la siguiente posición libre.

Modifica el programa del ejercicio 25 para añadir todos los alumnos del array a la clase Curso.

EJERCICIO 31

Buenas noticias!! Nos vamos a olvidar ya del alumno, el curso y todas esas clases. Cambiamos de tercio…

PARTE 1

Vamos a crear una clase Fecha que nos permita trabajar con este tipo de dato. Piensa un diseño para la clase, es decir, piensa que atributos y métodos debería de tener. Ponte en el lugar de una empresa que está desarrollando esta clase y su objetivo luego es utilizarla en todos sus programas o incluso venderla para que la usen otras empresas.

No implementes aún nada, sólo piensa en el diseño que harías. Luego, ponlo en común en clase.

PARTE 2

Implementa el diseño que habéis decidido de mutuo acuerdo entre todos. Crea un programa que utilice la clase Fecha.

EJERCICIO 32

Construye una nueva clase Reloj que debe funcionar del siguiente modo:

1) Tendrá estos atributos:
    a. Horas, Minutos, Segundos → Integer
    b. Formato24horas → Boolean
2) Tendrá dos constructores:
    a. Por defecto: Inicializará a las 00:00:00 los valores de horas, minutos, segundos. Y a TRUE el formato24horas
    b. Otro que reciba los parámetros horas, minutos, segundos. Pondrá por defecto a TRUE el formato24horas
3) Tendremos métodos get() para todos los atributos
4) Tendremos dos métodos ponerEnHora:
    a. Uno sólo recibe horas y minutos. (Los segundos no se cambiarían)
    b. Otro que recibe horas, minutos y segundos
5) Tendremos un método set() para el atributo formato24Horas

6) Tendremos un método check() que devuelve true si la hora es correcta:
    a. Las horas tienen que estar entre 0 y 23
    b. Los minutos tienen que estar entre 0 y 59
    c. Los segundos tienen que estar entre 0 y 59
7) Tendremos el método toString() que devuelva la hora en dos formatos:
    a. 00:00:00 (si formato24Horas es true)
    b. 00:00:00 pm/am (si formato24Horas es false) Ten en cuenta que aquí tendrás que restar 12 a las horas para mostrarlos correctamente.
    c. Si la hora no es correcta (compruébalo llamando al método check()), entonces se mostrará "HORA INCORRECTA"
8) Tendremos un método equals() que indique que dos relojes son iguales si sus horas, minutos y segundos son iguales.

Crea un programa que use la clase Reloj del siguiente modo. Después de cada paso, imprime el reloj:

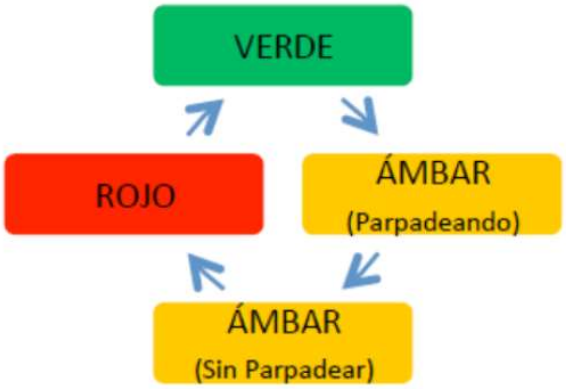
1. Crea un objeto reloj con el constructor por defecto
2. Solicita al usuario una hora correcta en tres pasos (hora, minutos, segundos). Y pon en hora el reloj.
3. Cambiar a formato NO 24 horas
4. Cambia la hora a las 24:17
5. Cambia la hora a las 21:82
6. Cambia la hora a las 17:16:15
7. Crea otro objeto reloj con el constructor que recibe la hora e indica esta: 17:16:15
8. Pregunta si los dos objetos son iguales e imprime el resultado

EJERCICIO 33

Construye una nueva clase Semaforo que debe funcionar del siguiente modo:

1) Tendrá dos atributos:
    a. color, de tipo String. Podrá ser Rojo, Ámbar o Verde. Lo correcto es que declares estos valores como constantes.
    b. parpadeando, de tipo Boolean.
2) Sólo tendrá el constructor por defecto, que tendrá que inicializar el color en Rojo sin parpadear.
3) Crea los métodos get y set
4) En los métodos set, debes validar lo siguiente:
    a. Cuando se intente cambiar el color, si se indica un color no válido, entonces no se modificará el atributo.
    b. Cuando se intente cambiar el estado de parpadeo, si se intenta activar el parpadeo cuando el color es distinto a Ámbar, entonces no se modificará.
5) Sobrescribe el método toString() para que devuelva lo siguiente en función del valor de los atributos:
    a. Semáforo en ROJO
    b. Semáforo en VERDE
    c. Semáforo en ÁMBAR
    d. Semáforo en ÁMBAR parpadeando

6) Añade un método cambiarEstado() que modifique el valor de los atributos del semáforo siguiendo este ciclo de estados:



Cuando lo tengas listo, crea un programa que siga la siguiente secuencia y demuestre el funcionamiento de la clase implementada. Después de cada paso, imprime el semáforo.

1. Crea un Semáforo.
2. Cambia el color a un color incorrecto.
3. Cambia el color a verde.
4. Pon el atributo parpadeando a cierto.
5. Cambia el color a ámbar.
6. Pon el atributo parpadeando a cierto.
7. Llama 5 veces al método cambiarEstado (usando un bucle)
8. Crea un nuevo semáforo que sea copia del anterior.

EJERCICIO 34

Crea un programa que solicite al usuario 5 cadenas y las meta en una lista (Hazlo con un bucle). Luego imprime la lista usando el método toString(). A continuación, haz lo siguiente:

1. Pon todas las cadenas de la lista en mayúsculas. Imprime la lista de nuevo.
2. Comprueba si la lista contiene alguna cadena vacía. En tal caso, escribe un mensaje por consola indicándolo. (No lo hagas con un bucle)
3. Borra todas las cadenas de la lista que tengan una longitud menor a 6 caracteres.

EJERCICIO 35

Vuele a hacer el ejercicio 25 utilizando List en lugar de arrays.

EJERCICIO 36

Vuele a hacer el ejercicio 30 utilizando List en lugar de arrays.

EJERCICIO 37

Una cola es una especie de lista donde entran y salen objetos. Cuando entra un objeto, se añade al final. Cuando sale un objeto, sale el que más tiempo lleve esperando dentro. Igual que cuando hacemos cola para entrar en algún lugar.

Crea la clase ColaCadenas que tenga estos dos métodos:

- añadirCadena(): recibe una cadena y no devuelve nada. Se tendrá que añadir esa cadena a la cola de espera.
- sacarCadena(): no recibe nada y devuelve una cadena. Tendrá que devolver la cadena que hace más tiempo que se añadió.
- getTamaño(): no recibe nada y devuelve un entero con todos los elementos que hay en la cola.
- toString(): debe imprimir el contenido de la cola

Implementa esta clase utilizando internamente un atributo de tipo List<String>.

Cuando la tengas lista, haz un programa para probarla con estos pasos (después de cada paso, imprime la cola):

1. Crea un objeto cola.
2. Añade a la cola las palabras “primero” y “segundo”
3. Saca de la cola un elemento e imprímelo.
4. Añade a la cola la palabra “tercero”
5. Saca todos los elementos que queden en la cola e imprímelos.
6. Añade a la cola la palabra cuarto.

EJERCICIO 38

Una pila es una especie de lista donde entran y salen objetos. Cuando entra un objeto, se añade encima de la pila. Cuando sale un objeto, sale el que menos tiempo lleve esperando dentro. Como si fuera una pila de platos o de documentos. El último que pongo encima será el primero que coja.

Crea la clase PilaCadenas que tenga estos dos métodos:

- añadirCadena(): recibe una cadena y no devuelve nada. Se tendrá que añadir esa cadena a la pila de espera.
- sacarCadena(): no recibe nada y devuelve una cadena. Tendrá que devolver la palabra que hace menos tiempo que se añadió.
- getTamaño(): no recibe nada y devuelve un entero con todos los elementos que hay en la pila.
- toString(): debe imprimir el contenido de la pila

Implementa esta clase utilizando internamente un atributo de tipo List<String>.

Cuando la tengas lista, haz un programa para probarla con estos pasos (después de cada paso, imprime la pila):

7. Crea un objeto pila.
8. Añade a la pila las palabras “primero” y “segundo”
9. Saca de la pila un elemento e imprímelo.

10. Añade a la pila la palabra “tercero”
11. Saca todos los elementos que queden en la pila e imprímelos.
12. Añade a la pila la palabra cuarto.

EJERCICIO 39 (Avanzado)

Crea una clase PilaColaCadenas que sea como las anteriores, pero pueda funcionar de un modo u otro. Para ello, tendrá dos métodos que sean setModoPila() y setModoCola() que no reciben ni devuelven nada, pero cambian el modo de funcionamiento.

El constructor pondrá por defecto el modo Cola cuando se cree un objeto.

EJERCICIO 40 (Avanzado)

Modifica la clase PilaColaCadenas para que se llame PilaCola y funcione con cualquier tipo de Objetos. El tipo de Objeto se indicará al crear la instancia de la clase, igual que hacemos cuando creamos un objeto Lista.

Pista: la clave está en la letra “E”

EJERCICIO 41

Vamos a crear un programa para gestionar los partidos de futbol de la liguilla interna de la CEU. Para ello, nos crearemos un conjunto de clases para poder manejar los datos. Las clases son las siguiente:

- Jugador.
  - Atributos privados con métodos get y set:
    - Nombre → String
    - Dorsal → Integer
  - Constructor que reciba los dos atributos por parámetro
  - Método equals() que compare los dorsales
  - Método toString() para que imprima algo tal que: “9 – Blas”
- Equipo.
  - Atributos privados con métodos get y set:
    - Nombre → String
    - Capitan → Jugador
    - Jugadores → Lista de Jugador
  - Constructor que reciba por parámetro el atributo nombre.
  - Método equals que compare los nombres.
  - Método toString para que se imprima algo tal que así:

    Real Madrid C.F. – Capitán: Marcelo – Jugadores: [1 – Courtois, 2- Carvajal, …]

- Resultado.
  - Atributos privados con métodos get y set:
    - golesLocales → Integer
    - golesVisitante → Integer
  - Constructor por defecto que inicializa el resultado a 0
  - Método equals que compare los dos atributos
  - Método toString para que se imprima algo tal que así: 0 – 0
  - Métodos:
    - isVictoriaLocal() que devuelve Boolean si han ganado los locales
    - isVictoriaVisitante() que devuelve Boolean si han ganado los visitantes
    - isEmpate() que devuelve Boolean si ha sido empate
- Partido.
  - Atributos privados con métodos get y set:
    - equipoLocal → Equipo
    - equipoVisitante → Equipo
    - resultado → Resultado
  - Constructor por defecto
  - Método toString para que se imprima algo tal que así:
    Real Madrid C.F. vs F.C.Barcelona [0 – 0]
  - Métodos:
    - getEquipoGanador() que devuelve el Equipo que haya ganado. Si hay empate, devolverá null.

Cuando tengas todas las clases terminadas, crea un programa que haga lo siguiente

1. Crea dos equipos con al menos 3 jugadores cada uno. Solicitando los datos al usuario. Designa al capitán de cada equipo (será el primer jugador indicado). Cuando los tengas, imprime los equipos por consola.
2. Crea un partido para estos dos equipos. Establece el resultado en 0 a 0. Imprime el partido.
3. Pregunta al usuario por el resultado y cámbialo. Imprime el partido.
4. Imprime el equipo ganador.
5. Añade al equipo visitante un jugador con el dorsal 99 y nombre “Blas infiltrado”. Imprime el equipo visitante.
6. Cambia el capitán del equipo local para que sea el último jugador de la lista de sus jugadores. Imprime el equipo local.

EJERCICIO 42

Añade a la clase Equipo del ejercicio anterior un atributo de tipo String que sea “competición”. Queremos que ese atributo tenga el mismo valor siempre para todos los Equipos que se creen. Añade métodos get() y set().

Modifica el programa luego para que, después del paso 6, haga lo siguiente:

7. Cambia la competición del primer equipo por “Liga Nacional”.
8. Imprime la competición del segundo equipo.

9. Cambia la competición del segundo equipo por “Copa Internacional”
10. Imprime la competición del primer equipo

(Si todo está correcto, en el paso 8 y en el 10 debería imprimirte siempre lo que se acaba de cambiar, aunque sea a través del otro objeto)

EJERCICIO 43

Nota: utiliza lo que has aprendido de herencia y de clases abstractas.

Necesitamos programar una aplicación para gestionar una base de datos de películas. De cada película recogeremos los siguientes datos (tendremos que tener métodos get() y set() para ellos):

- Año de estreno
- Título
- Lista de actores
- Su guionista
- Su director

Tanto actores como guionistas y directores se identifican por su nombre, año de nacimiento y nacionalidad. Cada una de estas clases tendrá que tener:

- Métodos get() y set() de todos esos atributos.
- Método getSueldo(), que es constante para cada tipo:
    - Los actores cobran 100.000 euros
    - Los guionistas cobran 50.000 euros
    - Los directores cobran 200.000 euros.

Crea las clases necesarias para gestionar esta aplicación. Cuando termines, realiza un programa que haga lo siguiente:

1. Crea 4 actores:
    a. Blas Blau de España nacido en 1983
    b. Laura Pozo de Italia nacida en 1981
    c. Marcel Cade de Suiza nacido en 2001
    d. Violeta Volo de Rusia nacida en 1999
2. Crea 1 directora:
    a. Sara Marea de Portugal nacida en 1994
3. Crea 2 guionistas:
    a. Marco Smith de Reino Unido nacido en 1988
    b. Cheng Shu de China nacido en 1977
4. Crea una película que se estrena en 2027 dirigida por Sara y con Marco como guionista. Actuarán Blas y Laura. Su título es “Do you know Joe Blas?”
5. Crea otra película con el mismo año de estreno y directora. El guionista es el chino, y participan todos los actores. Su título es “Muerte en la sombra”
6. Imprime la lista de actores de la primera película
7. Imprime el sueldo del guionista de la primera película y su nacionalidad
8. Borra al actor suizo de la segunda película y añádelo a la primera
9. Imprime los actores de las dos películas

EJERCICIO 44

Tenemos que programar un video juego que permita que un jugador pueda crear equipos de combate y competir con otro jugador en una partida. De cada jugador registraremos su nombre y su equipo de combate (sólo puede tener uno). Estos equipos se componen de dos tipos de personajes:

- Asesinos (Tienen 100 puntos de vida, y quitan 10 puntos de vida al rival cada 5 segundos)
- Parásitos (Tienen 200 puntos de vida, y quitan 2 puntos al rival cada segundo)

Todos los personajes tienen un nombre y un código. Se pueden llegar a tener en el equipo un máximo de 5 personajes. Y no se podrán tener personajes repetidos. El equipo tendrá además un número de puntos de vida. Estos puntos son el resultado de la suma de puntos de vida de cada uno de los personajes.

Crea las clases necesarias para gestionar el programa. Como mínimo: Jugador, EquipoCombate, Asesino, Parásito. Crea métodos get() y set() que consideres. Y añade algún método más si lo crees necesario para que todo funcione correctamente. Todas las clases tendrán método toString() completo.

Después, haz un programa que haga lo siguiente:

1. Crea un jugador que tenga un equipo de combate compuesto por 3 asesinos.
2. Crea un parásito con nombre “Blas” y código “B69”. Muestra cuántos puntos quita al rival y cada cuánto tiempo.
3. Añade el parásito Blas al jugador.
4. Imprime los puntos de vida del equipo de combate del jugador
5. Intenta añadir de nuevo el parásito Blas al equipo del jugador
6. Imprime todos los personajes del equipo de combate del jugador (Deberían ser sólo 4)
7. Crea otros dos parásitos nuevos. Intenta añadirlos al equipo de combate del jugador.
8. Imprime todos los personajes del equipo de combate del jugador (Deberían ser sólo 5)
9. Imprime los puntos del equipo de combate del jugador. (Deberían ser 700)

.

EJERCICIO 45

Crea un programa que solicite al usuario su fecha de nacimiento. A continuación, usando la clase LocalDate, muéstrale la siguiente información:

1. Si la fecha no es correcta, vuelve a solicitarla hasta que lo sea.
2. Dile si nació o no en un año bisiesto.
3. Dile el día de la semana en el que nació.
4. Dile cuántos años tiene.
5. Dile, suponiendo que va a vivir exactamente 100 años, cuánto tiempo de vida le queda (días, meses y años)
6. Dile, ya de regalo, que hora es ahora en formato hora/minuto/segundo con 24 horas. Por ejemplo: 17:23:55

EJERCICIO 46

Implementa la interfaz Fecha del ejercicio 31 utilizando en tu clase un atributo de la clase LocalDate

EJERCICIO 47

Nos han encargado que programemos unas clases para gestionar el Carrito de compra de una tienda online. El carrito tendrá que guardar esta información:

- Fecha de creación
- Fecha de última actualización (se debe actualizar cada vez que se haga un cambio en la cesta)
- Cliente
  - DNI
  - Nombre
- Lista de artículos

Los artículos los hay de dos tipos: Ropa y Libros. Todos tienen una descripción y un precio. Pero la ropa además tiene talla y color. Y los libros tienen autor.

Nos piden que el carrito tenga estos métodos:

- get() de todos los atributos
- Constructor único que recibirá un Cliente. Debe inicializar la fecha de creación y de última actualización con la fecha del momento en el que se crea el carrito.
- getCantidad() que devuelve la cantidad total de artículos en la cesta
- getTotal() que devuelve el importe total que habría que pagar
- getPrecioMedio() que devuelve el precio medio de la cesta (división del total entre la cantidad de artículos)
- toString() que imprima datos de cliente, cantidad de artículos, suma del total a pagar y fecha de última actualización en formato dd/MM/yyyy
- addArticulo() que recibe un artículo y lo añade a la lista
- borrarArticulo() que recibe un entero y borra el artículo que ocupa esa posición.
- vaciarCesta() que elimina todo el contenido de la cesta.

Realiza un programa que haga lo siguiente:

1. Crea un cliente con dni 12345678X y nombre “Blas de los Montes”.
2. Crea una cesta para el cliente anterior. Imprímela
3. Crea un artículo de tipo Ropa que cueste 20 euros, se llame “Poncho”, con talla XL y color azul.
4. Añade el artículo anterior dos veces a la cesta. Imprímela
5. Crea un artículo de tipo Libro que cueste 15 euros, se llame “Así habló Zaratustra” del autor “Nietzsche”
6. Añade el artículo anterior una vez a la cesta. Imprímela
7. Borra el artículo que aparece en la posición 1 de la cesta. Imprímela
8. Obtén e imprime el precio medio del artículo.
9. Vacía la cesta e imprímela.
10. Obtén e imprime el precio medio del artículo.

EJERCICIO 48

Cambia el ejercicio anterior para que no se puedan meter artículos iguales en la cesta. Utiliza la interfaz Set. Revisa cómo cambiaría el programa que hemos hecho. ¿Los importes de la cesta son iguales antes de vaciarla?

EJERCICIO 49

Repite el ejercicio 34 utilizando un Set en lugar de una lista.

EJERCICIO 50

Crea una clase Pais con dos atributos:

- Código (será el ISO_3166-1 https://es.wikipedia.org/wiki/ISO_3166-1)
- Descripción

Añade:

- Métodos get() y set()
- toString() para mostrar los dos atributos
- Constructor único que reciba los dos atributos
- Método equals() que compare el atributo código (no borres el hashCode())

Crea un programa que haga lo siguiente:

1. Crea un Set vacío e imprímelo
2. Crea tres países:
    a. ES / España
    b. CK / Islas Cook
    c. CK / Islas Caimán
3. Añade los 3 países al conjunto e imprímelo (Deberían aparecer 2. ¿Sabes por qué?)
4. Crea otro país (es / España) y añádelo al conjunto. Imprímelo.
5. ¿Qué podrías hacer para que no ocurriera lo del punto anterior (el que se puedan añadir países al conjunto que son iguales a otro, aunque el código esté en minúsculas)? Si sabes qué… hazlo y repite el punto anterior.
6. Borra todos los países que no sean “ES” del conjunto. Luego imprímelo.
7. Vacía el conjunto.

EJERCICIO 51

Crea un programa que haga lo siguiente:

1. Solicita al usuario 5 números BigDecimal y los metes en una lista. Cada número que te proporcione tiene que ser mayor al anterior, si no es así, vuelve a solicitarlo.
2. Cuando termines, imprime la suma de todos los números. La suma debe estar redondeada a 1 decimal con HALF_DOWN
3. Luego imprime la división del primer número entre el segundo, redondeando el resultado a 3 decimales con HALF_UP.

EJERCICIO 52

Cambia el ejercicio 47 para que el precio de los artículos sea un BigDecimal. Adapta lo que consideres necesario.

EJERCICIO 53

Crea una clase Hucha que nos permita gestionar nuestros ahorros. Tendrá que registrar únicamente el importe que tenemos actualmente ahorrado. Usa la clase BigDecimal.

Cuando se crea una hucha nueva estará inicialmente vacía, como es lógico.

El importe que tenemos en la hucha siempre estará redondeado a 2 decimales.

Tendremos los siguientes métodos:

- meterDinero() que recibe un BigDecimal con el importe a introducir y nos devuelve otro BigDecimal con el importe que quedará en la Hucha
- sacarDinero() que recibe un BidDecimal con el importe a sacar y nos devuelve otro BigDecimal con el importe finalmente sacado. Si se intenta sacar más dinero del que hay, sólo se sacará lo que haya.
- contarDinero() que nos devuelve un BigDecimal con el importe que tenemos en la Hucha.
- romperHucha() saca todo el dinero de la Hucha y nos devuelve un BigDecimal con todo lo sacado.
- toString() que imprime el importe que hay en la Hucha con separadores decimales y de miles, siempre con dos decimales rellenos y al menos un dígito entero, y el símbolo del euro. Ejemplos: “0,35 €”, “123,40€”, “1.123,00 €”

Cuando tengas la clase, crea un programa que haga lo siguiente (después de cada paso, imprime siempre la hucha):

1. Crea una hucha. Debe imprimir 0,00 €
2. Mete 100 euros. Debe imprimir 100,00 €
3. Mete 50,501 euros. Debe imprimir 150,50 €
4. Saca 20,5 euros. Debe imprimir 130,00 €
5. Intenta sacar 200 €. Debe imprimir 0,00 €
6. Vuelve a meter en la hucha el importe que habías sacado. Debe imprimir 130,00 €
7. Llama a contarDinero(). Lo que devuelva, mételo en la hucha. Debe imprimir 260,00 €
8. Llama a romperHucha(). Lo que devuelva, mételo en la hucha de nuevo. Debe imprimir 260,00 €

EJERCICIO 54

(Utiliza LocalDate, List y BigDecimal donde corresponda)

Crea una clase CuentaAhorros. Esta cuenta bancaria tendrá un número de 20 dígitos que podrá empezar por cero y una lista de movimientos bancarios.

Todos los movimientos tendrán registrada su fecha. Los movimientos podrán ser de 3 tipos:

- Cargo (llevan un importe y un CIF de la empresa que hace el cargo). Los cargos se imprimen así:
  [C - Fecha - Importe - CIF]. Por ejemplo: [C - 18/02/2022 - 33,21 € - 98765432F]
- Ingresos (llevan un importe y una descripción). Los ingresos se imprimen así:
  [I - Fecha - Importe - descripción]. Por ejemplo: [I - 18/02/2022 - 3,21 € - Ejemplo]
- Retiradas (llevan un importe). Las retiradas se imprimen así:
  [R - Fecha - Importe]. Por ejemplo: [R - 18/02/2022 - 12,00 €]

Una vez registrado un movimiento, este no se podrá borrar ni modificar.

Necesitamos estos métodos:

- Constructor de la clase CuentaAhorros que reciba el número de cuenta
- Método para añadir un movimiento a la cuenta
- Método para obtener el total de dinero en la cuenta
- Método para obtener una lista de cadenas que muestre todos los movimientos de la cuenta.
- Método igual que el anterior pero que sólo devuelva las retiradas
- Método igual que el anterior pero que sólo devuelva los ingresos
- Método igual que el anterior pero que sólo devuelva los cargos

Crea un programa que haga lo siguiente:

1. Crea una cuenta corriente e imprime el dinero que tenemos
2. Añade 2 cargos, 2 ingresos y 1 retirada
3. Imprime el dinero que tenemos
4. Imprime todos los movimientos
5. Imprime los cargos
6. Imprime los ingresos
7. Imprime las retiradas

EJERCICIO 55

Necesitamos un programa para registrar las notas de todos los alumnos. Para ello, crea una clase que se llame Evaluacion que tenga dentro un mapa de String → BigDecimal

La clase tendrá estos métodos:

1. addNota() que reciba el dni del alumno, su nota, y lo añada al mapa. Si ya tenemos registrada una nota para el mismo alumno, no haremos nada (no sobrescribimos). El método debe devolver un booleano indicando si finalmente hemos añadido o no algo al mapa.
2. corregirNota() que reciba el dni del alumno y su nota. Si ya tenemos para el alumno una nota, la cambiamos. Si no tenemos ninguna, no hacemos nada. El método debe devolver un booleano indicando si finalmente hemos corregido algo o no en el mapa.
3. obtenerNotaAlumno() que recibe un dni y devuelve la nota de ese alumno. Si no la tuviéramos registrada, devolveremos un 0.
4. obtenerNotaMedia() que no recibe nada y devuelve un BigDecimal con la nota media calculada de todos los alumnos.
5. obtenerCantidadAprobados() que devuelva un entero con la cantidad de alumnos que tengan al menos un 5
6. obtenerSuspensos() que devuelva una lista con todos los DNIs de los alumnos suspensos
7. borrarAprobados() que borre todos los alumnos del mapa que estén aprobados.
8. toString() que imprima el listado con este formato:

```
Aprobados:
     2332323D (7,2)
     6332363F (5,2)
     0912399Z (9,1)
Suspensos:
     7561323B (3,2)
     2913853R (4,9)
```

Luego, realiza un programa que use la clase anterior:

1. Crea un objeto evaluación
2. Introduce 5 notas de distintos valores e imprime la evaluación
3. Intenta volver a introducir una nota para un alumno repetido. Comprueba que no se modifique imprimiendo la evaluación de nuevo.
4. Intenta corregir dos notas, una de un alumno que esté ya registrado y otra de un alumno no registrado. Comprueba que funciona correctamente imprimiendo de nuevo la evaluación.
5. Obtén la nota de algún alumno y la nota media de todos. Imprímelas formateando correctamente los decimales
6. Obtén e imprime la cantidad de aprobados
7. Obtén e imprime la lista de suspensos
8. Borra los aprobados. Comprueba que funciona bien volviendo a imprimir la evaluación.

CEU Andalucía
Abel Morillo Sánchez

EJERCICIO 56

Haz un programa que solicite números enteros al usuario hasta que nos indique el 0.

El programa tendrá que ir contando cuántas veces nos ha indicado cada número. Por ejemplo, imaginemos que el usuario nos va diciendo estos números:

3, 6, 8, 3, 2, 3, 4, 6, 3, 8, 0

El programa, al terminar, tendrá que imprimir algo así:

```
Total números indicados: 10
Distribución:
      > Número 3: 4 veces
      > Número 4: 1 vez
      > Número 6: 2 veces
      > Número 8: 2 veces
      > Numero 2: 1 vez
```

PISTA: Utiliza un Map donde las keys son los números que nos van indicando y los values son la suma acumulada de ese número.

EJERCICIO 57

Haz un programa que solicite palabras al usuario y las almacene hasta que escriba FIN.

Después, pide al usuario que te de alguna letra y le muestras todas las palabras almacenadas que empiecen por esa letra. Así continuamente hasta que el usuario vuelva a escribir FIN.

Por ejemplo, imaginemos que el usuario nos dice estas palabras en la primera parte:

Blas, Sevilla, regadera, Sol, bota, FIN

La segunda parte del programa se ejecutaría así:

```
Dime una letra: B
Hay 2 palabras que empiezan por B:
      > Blas
      > bota

Dime una letra: T
No hay palabras que empiezan por T.

Dime una letra: R
Hay 1 palabra que empieza por R:
      > regadera

Dime una letra: FIN
¡Gracias por jugar con nosotros! Bye
```

PISTA: Utiliza un Map donde la clave sea un String (la letra) y el valor sea una List<String> (las palabras). Es la última pista relativas a Map que te doy…

EJERCICIO 58

Imagino que habrás hecho el ejercicio anterior con una única clase todo apiñado en el método main… como si acabaras de entrar al curso. Me has defraudado ☹

Modifícalo para que quede un programa más presentable:

- Crea una clase diccionario que tenga como atributo el mapa
- Añade un método que se llame cargarDiccionario() que reciba una List<String> con la lista de palabras que se quieren cargar (habrá palabras que empiecen por diferentes letras)
- Añade otro método que sea borrarDiccionario() y borre todo
- Añade otro método que sea imprimirPalabras() que reciba un String con la letra que queremos imprimir.

En general, aplica siempre un divide y vencerás. Intenta dividir tus problemas en clases y métodos.

EJERCICIO 59

Crear una clase que se llame Geografia que contenga un atributo que sea un mapa llamado paises de String → String, donde la clave es el nombre del país y el valor su capital. Tendrá los métodos:

1. agregarPais. Recibe un país y su capital y lo añade al mapa. El país lo guardará siempre en mayúsculas y la capital la guardará con la primera letra sólo en mayúsculas. Si el país ya está en el mapa, actualizará la nueva capital. Devolverá true si el país estaba en el mapa y false si no estaba.
2. obtenerCapital. Recibe un país y devuelve la capital de dicho país. En caso de que no exista el pais en el mapa, devolverá la cadena vacía. OJO: el país puede que lo recibas en minúsculas.
3. imprimirPaises: imprime el mapa de la siguiente manera: PAIS: “país” – CAPITAL: “capital” (uno en cada línea)
4. eliminarPais: Recibe un país y borra la entrada del mapa de dicho país si existe. OJO: el país puede que lo recibas en minúsculas.
5. calcularLongitudMediaPaises. Devuelve la media del número de caracteres que tiene el nombre de los países. Es decir, la suma total de caracteres de todos los nombres de los países entre la cantidad. Si no tenemos países, debe devolver 0.
6. eliminarPaisConCapitalLetra: Recibe una letra, y elimina del mapa la entrada donde la capital comience por letra especificada. Eliminará sólo la primera ocurrencia del mapa que coincida.
7. numeroPaisesConCapitalLetra: Recibe una letra y devuelve el número de países cuya capital comience con la letra especificada.
8. imprimirNumeroPaisesLetra: Recibe una letra e imprimirá las entradas del mapa cuya capital empiece por la letra especificada. Si no hay ninguna, imprimir: “Ninguna capital del mapa comienza por (”letra”). Suponemos que la letra recibida no está vacía ni es null.

9. mismaLetra: No recibe nada. Imprime las entradas del mapa cuyo nombre y su capital comiencen por la misma letra. Si no hay ninguna, imprimirá: no hay ningún país y capital que comiencen por la misma letra.

Crea un programa que pida 5 países con capitales y las guarde en el mapa. Si el país ya estaba insertado no cuenta como uno de los 5.

Ejemplo de prueba:

Egipto → El Cairo

Suiza → Berna

Austria → Praga

Hungría → Budapest

Austria → Viena

Brasil → Brasilia

A continuación:

- Imprime el mapa completo.
- Calcula el promedio del número de letras que contienen los países.
- Solicita una letra y muestra los países cuya capital empiezan por dicha letra: Ejemplo B
- Muestra los países cuyo nombre y capital comiencen por la misma letra.
- Elimina el primer país que encuentre cuya capital comience por la letra solicitada anteriormente.
- Vuelve a mostrar el número de países con capitales que comiencen por dicha letra.