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 cumplan lo anterior.

Los atributos se pueden escribir y leer públicamente.
Sobreescribe el método ToString para que muestre toda la información de la cuenta.

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.

En el método Main crea 2 cuentas, la primera indicando solamente el nombre y la segunda indicando nombre y una cantidad. Ingresa dinero en cada cuenta, luego retira dinero de ambas cuentas. Finalmente, muestra ambas cuentas por pantalla a través de sus métodos ToString.

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

- Sus atributos son: nombre, edad, dni, sexo (H hombre, M mujer), peso (kg) y altura(metros). No queremos que se accedan directamente a ellos.
- 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.
- Propiedades de escritura pública para todos los atributos menos dni.
- 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 implementarán son:
    - CalcularIMC(): calculará si la persona está en su peso ideal (peso en kg/(altura^2 en metros)), 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. Usa constantes para devolver estos valores.
    - EsMayorDeEdad(): indica si es mayor de edad, devuelve un booleano.
    - ToString(): devuelve toda la información del objeto.
    - GeneraDNI(): devuelve un string. 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.

El programa hará 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 las propiedades 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 persona.

Para calcular la letra, cogeremos el resto de dividir nuestro dni entre 23, el resultado debe estar entre 0 y 22. Haz un método donde según el resultado de la anterior fórmula busque en un array de caracteres la posición que corresponda a la letra. Esta es la tabla de caracteres:

RESTO	0	1	2	3	4	5	6	7	8	9	10	11
LETRA	T	R	W	A	G	M	Y	F	P	D	X	B

RESTO	12	13	14	15	16	17	18	19	20	21	22
LETRA	N	J	Z	S	Q	V	H	L	C	K	E

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 siguiente:
    - Un constructor por defecto. Generará una contraseña aleatoria con la longitud 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(): devuelve un string. Genera la contraseña del objeto con la longitud que tenga. No será visible para exterior.
- Propiedad pública get para contraseña y longitud.
- Propiedad pública set para longitud.

Ahora, en el método Main:

- Crea un array de Passwords con el tamaño que tu le indiques por teclado.
- Indica también por teclado la longitud de los Passwords
- Crea un bucle que cree un objeto para cada posición del array.
- 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: contraseña valor_booleano
  Ejemplo:
  123456 False
  abcdEFA123456 True

Ejercicio 4.

Crea una clase llamada Electrodoméstico 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 será blanco, el consumo energético será F, el precio base es de 100 € y el peso de 5 kg. Usa constantes para ello.
- Los colores disponibles son blanco, negro, rojo, azul y gris.
- Los constructores que se implementarán serán
    - Un constructor por defecto.
    - Un constructor con el precio y peso. El resto por defecto.
    - Un constructor con todos los atributos.
- Crea propiedades públicas de lectura para los atributos.
- Los métodos que implementara serán:
    - ComprobarConsumoEnergetico(letra): comprueba que la letra es correcta, sino es correcta usará la letra por defecto. Se invocará al crear el objeto y no será visible.
    - ComprobarColor(color): comprueba que el color es correcto, sino lo es usa el color por defecto. Se invocará al crear el objeto y no será visible.
    - PrecioFinal(): según el consumo energético, aumentará 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 clase llamada Lavadora con las siguientes características:

- Hereda de Electrodomestico
- 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 implementarán 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.
- Propiedad de lectura pública para el atributo carga
- Los métodos que se implementara serán:
- PrecioFinal(): si tiene una carga mayor de 30 kg, aumentará el precio 50 €, sino es así no se incrementará 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 clase llamada Television con las siguientes características:

- Hereda de Electrodomestico
- Sus atributos son resolución (en pulgadas) y sintonizador TDT (booleano), además de los atributos heredados.
- Por defecto, la resolución será de 20 pulgadas y el sintonizador será false.
- Los constructores que se implementarán 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:
- Propiedades de lectura pública para los atributos resolución y sintonizador TDT.
- PrecioFinal(): si tiene una resolución mayor de 40 pulgadas, se incrementará el precio un 30% y si tiene un sintonizador TDT incorporado, aumentará 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 is.
- 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.
Crea una clase llamada Persona. Sus atributos son: nombre, edad y dni. Esta clase contiene:

- Un constructor con todos sus atributos como parámetros el cual inicializará los datos. El dni se introduce sin letra.
- Setters y getters públicos para cada uno de los atributos. A la hora de guardar el dni hay que calcular la letra correspondiente y añadirla antes de guardarlo.
- ToString(): Devuelve todos los datos de la persona.
- EsMayorDeEdad(): Devuelve un valor lógico indicando si es mayor de edad.

Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular (que es una persona) y cantidad (puede tener decimales). A la hora de crear una instancia, el titular será obligatorio y la cantidad es opcional. Esta clase contiene:

- Dos constructores.
- Setters y getters públicos para cada uno de los atributos.
- ToString(): Devuelve todos los datos de la cuenta.
- Ingresar(cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- Retirar(cantidad): se retira una cantidad a la cuenta. La cuenta puede estar en números rojos.

Crea una nueva clase CuantaJoven que deriva de la anterior. Cuando se crea esta nueva clase, además del titular y la cantidad se debe guardar una bonificación que estará expresada en tanto por ciento. Construye lo siguiente para la clase:

- Dos constructores para indicar en cada uno de ellos la bonificación. hay que respetar que el titular es obligatorio y la cantidad opcional.
- Setters y getters públicos para cada uno de los atributos.
- En esta ocasión los titulares de este tipo de cuenta tienen que ser mayor de edad., por lo tanto hay que crear un método EsTitularValido() que devuelve verdadero si el titular es mayor de edad pero menor de 25 años y falso en caso contrario.
- Además la retirada de dinero sólo se podrá hacer si el titular es válido.

- El método ToString() debe devolver el mensaje de “Cuenta Joven”, la bonificación de la cuenta y los valores de los atributos heredados.
- Piensa los métodos heredados de la clase padre que hay que sobreescribir.

En el método Main:
- Crea dos personas, María con 20 años y dni 123456789 y Pedro con 40 años y dni 987654321.
- Crea una cuenta con Pedro y dos cuenta joven, una con María y otra con Pedro.
- Ingresa y retira dinero de las tres cuentas.
- Muestra por pantalla la información de las personas y de las cuentas.

Ejercicio 6.
Desarrolla una clase Cancion con las siguientes características:
- Atributos:
    - título: una variable string que guarda el título de la canción.
    - autor: una variable string que guarda el autor de la canción.

- Propiedades de lectura pública para los atributos.

- Un constructor que recibe como parámetros el título y el autor de la canción (por este orden).

Desarrolla una clase CompactDisc con las siguientes características:
- Atributos:
    - canciones: un array de objetos de la clase Cancion.
    - contador: la siguiente posición libre del array canciones.

- Un constructor en el cual se indicará el tamaño del array de canciones, contador se inicializa a 0.

- Propiedades:
    - NumeroCanciones: devuelve el valor del contador de canciones.

- Métodos:
    - ObtenerCancion(int): devuelve la Cancion que se encuentra en la posición indicada.
    - Agregar(Cancion): agrega en el siguiente hueco libre la Cancion proporcionada. Se colocará en canciones la canción pasada por parámetro en la posición que indique contador, luego se incrementa contador en 1. Si se intenta sobrepasar el límite de capacidad del CD no se agregará nada y se debe notificar al usuario.
    - Eliminar(): elimina la última Cancion añadida. Si no hay canciones no se elimina nada y se notifica al usuario. A la hora de eliminar hay que decrementar en 1 contador para que apunte al próximo hueco libre.

En el método Main:
- Crea tres canciones:
    - Take on me de A-ha.

    - Africa de Toto
    - Don't stop believin de Journey

- Crea un CD con un tamaño de 2 canciones.
- Graba la primera canción creada en el cd.
- Muestra por pantalla el número de canciones
- Borra una canción.
- Muestra por pantalla el número de canciones
- Agrega las 3 canciones al cd
- Muestra por pantalla el número de canciones
- Muestra por pantalla el título y el autor de la canción en la primera posición del cd.

Ejercicio 7.
Crea una clase llamada Serie con las siguientes características:

- Sus atributos son titulo, numero de temporadas, entregado, genero y creador.
- Por defecto, el numero 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 titulo y creador. El resto por defecto.
    - Un constructor con todos los atributos, excepto de entregado.
- Propiedades públicas get de todos los atributos, excepto de entregado.
- Propiedades públicas set de todos los atributos, excepto de entregado.
- Sobrescribe el método ToString para que devuelva toda la información de la serie.

Crea una clase Videojuego con las siguientes características:

- Sus atributos son titulo, horas estimadas, entregado, genero y compañia.
- 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 titulo y horas estimadas. El resto por defecto.
    - Un constructor con todos los atributos, excepto de entregado.
- Propiedades públicas get de todos los atributos, excepto de entregado.
- Propiedades públicas set de todos los atributos, excepto de entregado.
- Sobrescribe el método ToString para que devuelva toda la información del videojuego.

Con el fin de reutilizar lógica común, haz una interfaz llamada IEntregable con los siguientes métodos:

- Entregar(): cambia el atributo entregado a true.
- Devolver(): cambia el atributo entregado a false.
- EsEntregado(): devuelve el estado del atributo entregado.
- CompareTo(object), compara las horas estimadas en los videojuegos y en las series el número de temporadas. Como parámetro que tenga un objeto, es necesario que implementes la interfaz IComparable. Recuerda el uso de los casting de objetos y comprobar valores nulos. Si el objeto que se pasa por parámetro es nulo se devolverá un número mayor que 0.

Implementa la interfaz en las clases Videojuego y Serie. Ahora el método Main 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 cuantas Series y Videojuegos hay entregados y muestra por pantalla el resultado.
- Por último, indica el Videojuego que tiene más horas estimadas y la serie con más temporadas. Muéstralos en pantalla con toda su información (usa el método ToString()).

Ejercicio 8.
Nos piden hacer una un programa que gestione empleados.
Los empleados se definen por tener:

- Nombre
- Edad
- Salario (número real)

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 con todos los atributos, getters y setters públicos y sobreescribe ToString (aprovechando la herencia) para devolver toda la información de las clases.

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

Las clases tendrán un método llamado AplicarPlus()(este será un método abstracto en la clase Empleado), que según en cada clase tendrá una implementación distinta. Este método 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 aplicará el plus.
- En repartidor, si tiene menos de 25 y reparte en la “zona 3”, este recibirá el plus.

Crea una clase ejecutable donde crees distintos empleados, le apliques el plus y muestre sus salarios tras haber aplicado el plus..

Ejercicio 9.
Un array es una colección de datos estática debido a que una vez que se crea el tamaño no se puede cambiar. Vamos a crear una clase para solventar esta limitación.

Crea una clase llamada Lista<T> la cual será genérica para un tipo y que contendrá lo siguiente:

- Atributos:
    - items: un array de tipo T donde se guardarán los elementos de la colección.
    - nextPosition: un entero que marca la siguiente posición libre en el array items.

- Constructores:
    - Un constructor donde se le indica la capacidad/tamaño con la que crea el array items e inicializa nextPosition a 0.
    - Un constructor por defecto donde crea el array items con un tamaño predefinido en una constante (10) e inicializa nextPosition a 0.

- Propiedades:
    - Capacity: público de solo lectura, devuelve el tamaño que tiene el array items.
    - Count: público de solo lectura, devuelve cuántos elementos tiene la lista (el valor de nextPosition).

- Métodos:
    - Get(int index): devuelve el elemento que se encuentra en la posición index del array de items.
    - Add(T item): añade item a la lista. Para añadir el elemento, primero se debe comprobar si hay hueco en el array de items (ver si nextPosition es igual a la Capacidad), si no hay hueco entonces items va a ser igual a un nuevo array pero con el tamaño anterior duplicado (por ejemplo si items era un array de tamaño 2 ahora será un array de tamaño 4), a este nuevo array hay que meterle todos los valores que tenía el array anterior en la misma posición (por ejemplo si anteriormente items era [1, 2] ahora será [1 ,2, 0, 0]). Luego, habiendo hueco o no, se pone item en la posición nextPosition del array de items. Finalmente se incrementa en 1 nextPosition.
    - Remove(T item): elimina la primera ocurrencia de item de la lista. Para ello busca la primera ocurrencia de item en items, si la encuentra entonces todo lo que hay a la derecha de la posición en items se mueve 1 posición a la izquierda, colocando el valor por defecto del tipo en el nuevo lugar disponible (nextPosition - 1) y se decrementa en 1 nextPosition. Por ejemplo, si la lista contiene [1, 2, 3, 4] y se desea eliminar 2 items pasará a contener lo siguiente [1, 3, 4]. Ten en cuenta que item puede ser nulo y por lo tanto borrará la primera ocurrencia nula.
    - ToString(): devuelve un string con la información de la lista. La información será la concatenación, separado por comas, de lo que devuelva el método ToString de cada uno de los elementos de items hasta la posición nextPosition, todo ello entre corchetes. Por ejemplo, si items contiene tres elementos y tiene una capacidad de 5 (1, 2, 3, 0, 0), el método devolverá [1, 2, 3].

En el método Main:

- Crea una Lista de enteros con una capacidad de 2.
- Añade a la lista creada 20 números aleatorios comprendidos entre 1 y 10 ambos inclusive.

- Muestra la lista por pantalla usando su método ToString.
- Muestra por pantalla el elemento que está en la posición 10 en la lista.
- Llama al método Remove de la lista para que borre los números del 1 al 10 (10 llamadas al método Remove en total).
- Muestra la lista por pantalla usando su método ToString.

Ejercicio 10.
Vamos a implementar las barajas española y francesa.

Crea dos enumerados, cada uno en un archivo distinto. Los enumerados son:
- PalosBarajaEspanola con los valores: Oro, Copas, Espadas y Bastos.
- PalosBarajaFrancesa con los valores: Diamantes, Picas, Treboles y Corazones.

Crea una clase genérica de un tipo llamada Carta donde el tipo genérico es un enumerado y que tenga las siguientes características:
- Atributos
    - numero (int): indica el número de la carta.
    - palo (T): indica el palo de la carta.

- Constructores:
    - Un constructor donde se le indica el número y el palo de la carta.

- Propiedades de lectura pública para los atributos.

- Métodos:
    - ToString(): devuelve un string con la información de la carta en formato <número> de <palo>, ejemplo 2 de Oro. A la hora de mostrar el número hay que mostrarlo según el nombre que tiene en su correspondiente baraja. En la baraja española el 1 es As, el 10 es Sota, el 11 es Caballo y el 12 es Rey. En la baraja francesa el 1 es As, el 11 es Jota, el 12 es Reina y el 13 es Rey.

Crea una clase Baraja<T> donde T será uno de los enumerados anteriormente creados de la cual no se podrá crear ninguna instancia y que tendrá las siguientes características:
- Atributos (todos protegidos)
    - cartas: un array de Carta<T> donde se van a almacenar todas las cartas de la baraja.
    - posicionSiguienteCarta: un entero que indicará la posición de la siguiente carta en la baraja.
    - numeroCartas: un entero donde se guardará el número de cartas que tiene la baraja.
    - cartasPorPalo: un entero donde se indica cuántas cartas por palo tiene la baraja.

- Constructores:
    - Un constructor por defecto donde inicializa el valor de posicionSiguienteCarta a 0.

- Métodos:
    - CrearBaraja(): es abstracto, público y no devuelve nada, servirá para crear cada tipo de baraja posteriormente y establecer los valores de los atributos cartas, numeroCartas y cartasPorPalo.
    - Barajar(): es público y lo que hace es ordenar de manera aleatoria todas las cartas de la baraja. A la hora de ordenar, por cada carta se hará un intercambio de posición, cambiando la posición de una carta por otra obtenida aleatoriamente. Finalmente la posición de la siguiente carta vuelve al inicio.
    - CartasDisponible(): devuelve cuantas cartas disponibles hay en la baraja.
    - CogerCarta(): devuelve la carta que se encuentra en la posicionSiguienteCarta. Luego incrementa en 1 posicionSiguienteCarta. Si no se puede sacar más cartas se muestra un mensaje por pantalla indicando al usuario que no hay más cartas y que vuelva a barajar, en este caso se devolverá nulo.
    - CogerCartas(int): devuelve tantas cartas como se indique por parámetro. Si se piden más cartas de las que hay en la baraja hay que notificar al usuario diciéndole que no se puede dar más cartas de las que hay. Si se piden más cartas de las que hay disponible se notifica al usuario diciéndole que no hay suficientes cartas disponibles. En esos dos casos se devolverá nulo. En otro caso se devuelven las cartas que se piden.
    - MostrarMonton(): muestra por pantalla todas las cartas que ya han salido. Si no ha salido ninguna se muestra un mensaje diciendo que no se ha sacado ninguna carta.
    - MostrarBaraja(): muestra las cartas que aún no han salido. Si han salido todas se muestra un mensaje diciendo que no hay cartas que mostrar.

Crea una clase BarajaEspanola que hereda de Baraja y que usará el palo PalosBarajaEspanola y que tendrá las siguientes características:

- Atributos
    - extendida: un booleano para indicar si hay que incluir el 8 y el 9.

- Constructores:
    - Un constructor en el que se indica si es extendida o no. Si es extendida el número total de cartas será 48 y 12 cartas por palo si no será 40 en total y 10 cartas por palo. Llamará al constructor padre y luego creará la baraja.

- Métodos:
    - CrearBaraja(): sobreescribe el método para que cree la baraja con las cartas usando el palo PalosBarajaEspanola. Ten en cuenta el número total de cartas y cuántas cartas hay por palo.

Crea una clase BarajaFrancesa que hereda de Baraja y que usará el palo PalosBarajaFrancesa y que tendrá las siguientes características:

- Constructores:
    - Un constructor por defecto. El número total de cartas será 52 y 13 cartas por palo. Llamará al constructor padre y luego creará la baraja.

- Métodos:
    - CrearBaraja(): sobreescribe el método para que cree la baraja con las cartas usando el palo PalosBarajaFrancesa. Ten en cuenta el número total de cartas y cuántas cartas hay por palo.
    - EsRoja(Carta): devuelve un booleano indicando si la carta tiene un palo de color rojo (Corazon o Diamante).
    - EsNegra(Carta): devuelve un booleano indicando si la carta tiene un palo de color negro (Trebol o Pica).

En el método Main:
- Crea 2 barajas, una española y otra francesa.
- En cada baraja:
    - Muestra por pantalla cuántas cartas disponibles hay en la baraja.
    - Coge una carta.
    - Coge 5 cartas
    - Muestra por pantalla cuántas cartas disponibles hay en la baraja.
    - Muestra por pantalla las cartas que se han sacado de momento.
    - Baraja la baraja.
    - Coge otras 5 cartas y muestra cada una por pantalla. Si la baraja es francesa indica si cada carta es roja o negra.
    - Muestra por pantalla las cartas que quedan en la baraja.

Ejercicio 11.
Crea una estructura llamada Vector3 con las siguientes características:
- 3 propiedades de tipo entero públicas de lectura y escritura para guardar las coordenadas (X, Y, Z).
- Un constructor por defecto que pondrá las tres propiedades un valor de 0.
- Un constructor para poder indicar los valores de las 3 coordenadas.
- Un método llamado Sumar el cual recibirá un Vector3 como argumento y devolverá un nuevo Vector3 con la suma de las coordenadas con el vector pasado por parámetro.
- Un método llamado Distancia el cual recibirá un Vector3 como argumento y devolverá la distancia euclídea (Pitágoras) entre los dos vectores.

Crea una clase llamada Jugador con las siguientes consideraciones:
- Un atributo de tipo Vector3 donde se guardará la posición, sin su propiedad.
- Un atributo que indica si está o no vivo, con su propiedad de lectura pública.
- Un constructor por defecto donde se sitúa la instancia en las coordenadas (0,0,0) y empieza vivo..
- Un constructor que recibe un Vector3 por parámetro para indicar la posición inicial, también empieza vivo.
- Un método público llamado Mover que recibe un Vector3 por parámetro que indica cuántas unidades se va a desplazar el Jugador, actualizando así su posición.
- Un método público llamado Morir, cuando se llama el jugador se muere.
- Un método público llamado Matar que recibe un Jugador por parámetro, si el jugador pasado por parámetro está a menos de 5 unidades de distancia respecto a la

posición de la instancia, entonces el jugador pasado por parámetro se muere. Usa constante para guardar la distancia.

En el método principal:

- Crea dos instancias de Jugador, jugador1 y jugador2.
- El primer jugador empieza en la posición (10, 0, 2) y el segundo en la posición (0, 0, 0).
- Simula una persecución mediante un bucle, el jugador2 va a perseguir al jugador1 hasta matarle. El jugador2 avanza a una velocidad de (2,0,0) por iteración y el jugador1 huye a una velocidad de (1, 0, 0). Finalmente muestra por pantalla cuántas iteraciones ha durado la persecución.