MÉTODOS

EJERCICIO 1

Diseña un método que admita una lista de valores numéricos decimales y devuelva el valor máximo.

Construye un programa que permita al usuario introducir dicha lista.

EJERCICIO 2

Diseña un método que lea un entero cualquiera y separe los dígitos en una lista, devolviéndola.

Construye un programa que permita al usuario introducir números.

EJERCICIO 3

Escribe un programa que muestre un menú que permita calcular un factorial o una potencia.

- El cálculo de dichas operaciones se realizará con sendas funciones.
- Solo se permitirá al usuario introducir número positivos, para ello, habrá que implementar un función que realice dicha tarea.

EJERCICIO 4

Implementa una función que calcule el número de Fibonacci de orden *n.*

EJERCICIO 5

Escribe una función *leeOpcion2Alternativas* que realice las siguientes tareas:

- Imprime en pantalla un mensaje.
- Lee una opción de forma que solo admite ‘s’, ‘S’, ‘n’, ‘N’.
- Devuelve la opción elegida. ¿Qué debería devolver un char o un boolean?.

Aplica esta función a un programa que lea si una persona es pensionista, autónomo o si está casado.

EJERCICIO 6

Diseña un método que, dado un conjunto de enteros, cuente el número de apariciones de un número dado.

EJERCICIO 7

Implementa un programa que compruebe si un número dado es feliz. Un número es feliz si sumando los cuadrados de sus dígitos, siguiendo dicho proceso, acabamos obteniendo uno (1) como resultado.

Por ejemplo, el número 203 es feliz ya que:

$$2^2 + 0^2 + 3^2 = 13 \rightarrow 1^2 + 3^2 = 10 \rightarrow 1^2 + 0^2 = 1$$

EJERCICIO 8

Implementa un método para obtener los valores de la función *f(x,y)* dentro de un intervalo de valores. Para ello, se deberá proporcional al método la dupla de valores (x,y) inferior y superior del intervalo, así como un valor k, el cual indicará el ratio de aumento de los valores del intervalo.

De esta forma, para la función $f(x,y) = \frac{\sqrt{x}}{y^2-1}$ y los valores (-50, -40); (50, 40) y k = 2; se calcularía los valores de la función en los puntos: (-50, -40), (-50, -38), (-50, -36), … (50, 34), (50, 36), (50, 38) y (50, 40).

EJERCICIO 9

Escribid un método que lea cuatro valores de tipo char (*min_izda*, *max_izda*, *min_dcha*, *max_dcha*) y devuelva en un array las parejas que pueden formarse con un elemento del conjunto {*min_izda* ... *max_izda*} y otro de {*min_dcha* ... *max_dcha*}. Por ejemplo, si *min_izda* = b, *max_izda* = d, *min_dcha* = j, *max_dcha* = m, el programa debe imprimir las parejas que pueden formarse con un elemento del conjunto {b, c, d} y otro de {j, k, l, m}

EJERCICIO 10

Diseña un método que lea un número entero *tope* y devuelva en un array los divisores de todos y cada uno de los números positivos menor o iguales que *tope*.

EJERCICIO 11

Dos números amigos son dos números naturales a y b, tales que la suma de los divisores propios de a más uno es igual a b, y viceversa. Un ejemplo de números amigos es el par de naturales (220,284), ya que:

- Los divisores propios de 220 son 2, 4, 5, 10, 11, 20, 22, 44, 55 y 110, que suman 283, y 283 + 1 = 284.
- Los divisores propios de 284 son 2, 4, 71 y 142, que suman 219, y 219 + 1 = 220.

Realiza un programa que implemente estas dos tareas:

a) En primer lugar debe leer dos números naturales e indicar si son o no amigos.
b) A continuación leerá otro número natural, n, e informará si existe algún número amigo de n en el intervalo centrado en n y de radio n/3.

EJERCICIO 12

Diseña un método el cual, pasándole un número entero cualquiera, indique si es primo o no.

EJERCICIO 13

Implementa un método que acepte cualquier número entero en base 10 y devuelva dicho número transformado a binario. No uses String.

EJERCICIO 14

Completa el ejercicio anterior para que pueda transformar un número entero de base 10 a hexadecimal. Ahora sí usa String. Añade también métodos para hacer la transformación contraria, es decir, de binario a base 10.

CLASES

EJERCICIO 1

Haz una clase llamada **Persona** que siga las siguientes condiciones:

- Sus atributos son: **nombre, edad, DNI, sexo** (H hombre, M mujer), **peso y altura**. Piensa qué 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 sera hombre por defecto, usa una constante para ello.
- Se implementarán 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 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, y la función devuelve un 0 y si devuelve un valor mayor que 25, significa que tiene sobrepeso y la función devuelve un 1.
  - **esMayorDeEdad()**: indica si es o no mayor de edad.

- **comprobarSexo(char sexo)**: comprueba que el sexo introducido es correcto. Si no es correcto, se pide otra vez. No será visible al exterior.
- **toString()**: devuelve toda la información del objeto en un solo String, separado por guiones.
- **generaDNI()[1]**: 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.

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.

[1]: Para asignar una letra al DNI sumaremos todos los números y haremos la operación módulo con el valor 23. Dependiendo del resultado asignaremos una letra en función de la siguiente tabla:

Tabla para calcular la letra del DNI

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 2

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.
- 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*** (la clase) 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 3

Crea una clase Fecha con atributos para el día, el mes y el año de la fecha.

Incluye, al menos, los siguientes métodos:

- Constructor predeterminado con el 1-1-1900 como fecha por defecto.
- Constructor parametrizado con día, mes y año.
- **leer()**: pedirá al usuario el día (1 a 31), el mes (1 a 12) y el año (1900 a 2050).
- **bisiesto()**: indicará si el año de la fecha es bisiesto o no.

- **diasMes(int)**: devolverá el número de días del mes que se le indique (para el año de la fecha).
- **valida()**: comprobará si la fecha es correcta (entre el 1-1-1900 y el 31-12-2050); si el día no es correcto, lo pondrá a 1; si el mes no es correcto, lo pondrá a 1; y si el año no es correcto, lo pondrá a 1900. Será un método auxiliar (privado). Este método se llamará en el constructor parametrizado y en leer().
- Getters y Setters.
- **corta()**: mostrará la fecha en formato corto (02-09-2003).
- **diasTranscurridos()**: devolverá el número de días transcurridos desde el 1-1-1900 hasta la fecha.
- **diaSemana()**: devolverá el día de la semana de la fecha (0 para domingo, ..., 6 para sábado). El 1-1-1900 fue domingo.
- **larga()**: mostrará la fecha en formato largo, empezando por el día de la semana (martes 2 de septiembre de 2003).
- **fechaTras(long)**: hará que la fecha sea la correspondiente a haber transcurrido los días que se indiquen desde el 1-1-1900.
- **diasEntre(Fecha)**: devolverá el número de días entre la fecha y la proporcionada.
- **siguiente()**: pasará al día siguiente.
- **anterior()**: pasará al día anterior.
- **copia()**: devolverá un clon de la fecha.
- **igualQue(Fecha)**: indica si la fecha es la misma que la proporcionada.
- **menorQue(Fecha)**: indica si la fecha es anterior a la proporcionada.
- **mayorQue(Fecha)**: indica si la fecha es posterior a la proporcionada.

En la clase ejecutable implementa los siguientes pasos:

- Pide al usuario el número de fechas que va a introducir.
- Construye un array de ***Fecha*** con tantas fechas como se haya indicado.

- Por cada ***Fecha*** pide al usuario el día, el mes y el año.
- Crea dos copias del array de ***Fecha***, fechas_asc, fechas_desc.
- Dentro del array fechas_asc, ordena las fechas de menor a mayor.
- Dentro del array fechas_desc, ordena las fechas de mayor a menor.
- Muestra los tres arrays.

EJERCICIO 4

Crea una clase ***Hora*** con atributos para las horas, los minutos y los segundos de la hora.

Incluye, al menos, los siguientes métodos:

- Constructor predeterminado con el 00:00:00 como hora por defecto.
- Constructor parametrizado con horas, minutos y segundos.
- **leer()**: pedirá al usuario las horas, los minutos y los segundos.
- **valida()**: comprobará si la hora es correcta; si no lo es la ajustará. Será un método auxiliar (privado) que se llamará en el constructor parametrizado y en leer().
- Getters y Setters.
- **print()**: mostrará la hora (07:03:21).
- **aSegundos()**: devolverá el número de segundos transcurridos desde la medianoche.
- **deSegundos(int)**: hará que la hora sea la correspondiente a haber transcurrido desde la medianoche los segundos que se indiquen.
- **segundosDesde(Hora)**: devolverá el número de segundos entre la hora y la proporcionada.
- **siguiente()**: pasará al segundo siguiente.
- **anterior()**: pasará al segundo anterior.
- **copia()**: devolverá un clon de la hora.
- **igualQue(Hora)**: indica si la hora es la misma que la proporcionada.

- **menorQue(Hora)**: indica si la hora es anterior a la proporcionada.
- **mayorQue(Hora)**: indica si la hora es posterior a la proporcionada.

En la clase ejecutable implementa los siguientes pasos:

- Implementa un array de tres posiciones de tipo ***Hora.***
- Pídele al usuario una hora e inicializa la primera posición del array con dicho dato.
- Crea una copia de dicho objeto para el resto de posiciones.
- Pídele al usuario una cantidad de tiempo concreta (3 horas y 2 minutos). Incrementa el segundo objeto del array dicha cantidad.
- Pídele al usuario otra cantidad de tiempo y decrementa el tercer objeto del array dicha cantidad.
- Ordena el array de mayor a menor.
- Muestra las horas.

EJERCICIO 5

Crear una clase Empleado que modele la información que una empresa mantiene sobre cada empleado: NIF, sueldo base, pago por hora extra, horas extra realizadas en el mes, tipo (porcentaje) de IRPF, casado o no y número de hijos.

La clase debe contemplar getters y setters para todos los atributos. Al crear los objetos se podrá proporcionar, si se quiere, el número de DNI. Los demás servicios que deberán proporcionar los objetos de la clase serán los siguientes:

- Cálculo y devolución del complemento correspondiente a las horas extra realizadas.
- Cálculo y devolución del sueldo bruto.
- Cálculo y devolución de las retenciones (IRPF) a partir del tipo, teniendo en cuenta que el porcentaje de retención que hay que aplicar es el tipo

menos 2 puntos si el empleado está casado y menos 1 punto (tanto por ciento) por cada hijo que tenga; el porcentaje se aplica sobre todo el sueldo bruto.

- **println()**: visualización de la información básica del empleado.
- **printAll()**: visualización de toda la información del empleado. La básica más el sueldo base, el complemento por horas extra, el sueldo bruto, la retención de IRPF y el sueldo neto.
- **copia()**: clonación de objetos.

En la clase ejecutable implementa los siguientes pasos:

- Pide al usuario el número de empleados que va a introducir.
- Construye un array de ***Empleado*** con tantos empleados como se haya indicado.
- Por cada ***Empleado*** pide al usuario sus datos.
- Calcula y muestra las retenciones de IRPF de cada empleado junto con sus datos.

EJERCICIO 6

Desarrolla una clase ***Canción*** con los siguientes atributos:

- **título**: una variable String que guarda el título de la canción.
- **autor**: una variable String que guarda el autor de la canción.

y los siguientes métodos:

- **Cancion(String, String)**: constructor que recibe como parámetros el título y el autor de la canción (por este orden).
- **Cancion()**: constructor predeterminado que inicializa el título y el autor a cadenas vacías.
- **dameTitulo()**: devuelve el título de la canción.
- **dameAutor()**: devuelve el autor de la canción.

- **ponTitulo(String)**: establece el título de la canción.
- **ponAutor(String)**: establece el autor de la canción.

EJERCICIO 7

Desarrolla una clase ***CD*** con los siguientes atributos:

- **canciones**: un array de objetos de la clase ***Canción***.
- **contador**: la siguiente posición libre del array canciones.

y los siguientes métodos:

- **CD()**: constructor predeterminado (creará el array canciones).
- **numeroCanciones()**: devuelve el valor del contador de canciones.
- **dameCancion(int)**: devuelve la ***Canción*** que se encuentra en la posición indicada.
- **grabaCancion(int, *Canción*)**: cambia la Cancion de la posición indicada por la nueva ***Canción*** proporcionada.
- **agrega(*Canción*)**: agrega al final del array la ***Canción*** proporcionada.
- **elimina(int)**: elimina la ***Canción*** que se encuentra en la posición indicada.

En la clase ejecutable implementa los siguientes pasos:

- Pide al usuario el número de CDs que va a introducir.
- Construye un array de ***CD*** con tantos CDs como se haya indicado.
- Por cada ***CD*** pide al usuario el número de canciones que va a incluir.
- Inicializa cada ***Canción*** del ***CD*** con sus datos correspondientes.
- Muestra una lista por cada ***CD*** con los datos completos de cada ***Canción***.

EJERCICIO 8

Crea una clase Libro que modele la información que se mantiene en una biblioteca sobre cada libro: título, autor (usa la clase Persona), ISBN, páginas, edición, editorial , lugar (ciudad y país) y fecha de edición (usa la clase Fecha).

La clase debe proporcionar los siguientes servicios: Getters y Setters, método para leer la información y método para mostrar la información. Este último método mostrará la información del libro con este formato:

```
Título: Introduction to Java Programming

3a. edición

Autor: Liang, Y. Daniel

ISBN: 0-13-031997-X

Prentice-Hall, New Jersey (USA), viernes 16 de
noviembre de

2001

784 páginas
```

EJERCICIO 9

Desarrollar una lista de ***Libros*** ordenada por título. La funcionalidad de la lista será la habitual: conocer el número de libros que hay en la lista, insertar un nuevo libro (en la posición que le corresponda), eliminar el libro de una determinada posición y obtener el libro de una determinada posición. También incluirá un método para buscar un libro a partir de una parte de su título (sin distinguir entre mayúsculas y minúsculas); el método devolverá la posición en la que se encuentra el libro (–1 si no se encuentra).

EJERCICIO 10

Desarrolla una aplicación ***Agenda*** que llevará las citas de una clínica dental. Esta ***Agenda*** será la correspondiente a un año concreto, estando esta compuesta por una serie de ***Páginas***. Cada ***Página*** estará dedicada a un día concreto del año, almacenando la ***Fecha*** concreta, así como una lista de las ***Citas*** que haya planificadas para ese día. En ***Citas*** se permitirá al usuario añadir la ***Hora***, así como un título y una descripción. ***Cita*** permitirá al usuario inicializarla con todos los datos necesarios, permitiendo modificar algún dato una vez creada, en caso de que fuese necesario. De igual forma, tendrá un método para mostrar los datos en un String.

Página permitirá al usuario añadir, borrar y ver todas las ***Citas*** que tenga almacenada. También permitirá buscar una ***Cita*** en base a la ***Hora*** proporcionada. La página se inicializará proporcionando la ***Fecha*** correspondiente.

Por último, ***Agenda*** se inicializará al proporcionarle un año en concreto. A partir de ese dato, se inicializará con tantas ***Páginas*** como días tenga dicho año. ***Agenda*** permitirá al usuario abrir una ***Página*** concreta (vendrá dada por la ***Fecha***).

La aplicación (main) tendrá un menú que permitirá al usuario inicializar una ***Agenda*** de un año concreto, y añadir, buscar y borrar citas de cualquier día. Se permitirá al usuario realizar estas acciones hasta que seleccione la opción de salir del programa.

EJERCICIO 11

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 quedará 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 está 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 cuántos 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 12

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 mostrará esa primera carta.

HERENCIA

EJERCICIO 1

Diseña una aplicación que gestione los productos de una tienda. Dichos 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 2

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 el 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.

El main tendrá un menú que ejecutará todas las funciones del almacén.

EJERCICIO 3

Vamos a hacer unas mejoras a la clase Baraja del ejercicio 12 de clases.

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

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

El método crearBaraja().

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 4

Implementa 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€

Existen 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.

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

Las clases tendrán un método llamado plus, que según la 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 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 y le apliques el plus para comprobar que funciona.

EJERCICIO 5

Implementa un programa que permita calcular el perímetro y el área de distintas figuras geométricas regulares.

Para ello, habrá de implementarse una clase FiguarRegular que contendrá los atributos de número de lados y longitud del lado.

Implementa también las clases Triángulo, Cuadrado y Pentágono las cuales heredarán de FiguraRegular.

Implementa una clase ejecutable con un array de 10 FigurasRegulares y calcula sus perímetros y áreas.