1. **Alumno**

Diseñar una clase llamada Alumno con siguientes atributos:

* nombre
* nota media

Y los métodos siguientes:

* Un **constructor** que inicialice los atributos con valores pasados como parámetros.
* Un método **get/set** para cada atributo
* Un método **toString**

Crear un programa Principal, que cree una instancia de un alumno y muestre los datos del alumno creado usando el método toString().

1. **Disco**

Diseñar una clase llamada Disco con siguientes atributos:

* autor
* titulo
* genero
* duración (duración total en minutos)

Y los métodos siguientes:

* Un **constructor** que inicialice los atributos con valores pasados como parámetros.
* Un método **get/set** para cada atributo
* Un método **toString**

Crear un programa Principal, que cree una instancia de un par de discos de distintos autores y muestre los datos creados usando el método toString().

1. **Piloto de Formula1**

Diseñar una clase llamada Piloto con siguientes atributos:

* nombre
* edad
* escuderia
* nacionalidad
* campeonatosGanados

Y los métodos siguientes:

* Un **constructor** que inicialice los atributos con valores pasados como parámetros.
* Un método **get/set** para cada atributo
* Un método **toString**

Crear un programa Principal, que cree una instancia para cada uno de los siguientes pilotos: Fernando Alonso, Lando Norris, Carlos Sainz y muestre sus datos.

1. **Gato**

Diseñar una clase llamada *Gato* que tenga en cuenta las siguientes características. Todos los gatos son de un color determinado, pertenecen a una raza, tienen una edad, tienen un determinado sexo - son machos o hembras - y tienen un peso que se puede expresar en kilogramos.

Para saber qué métodos debemos implementar hay que preguntarse qué acciones

están asociadas a los gatos. Bien, pues los gatos maúllan, ronronean, comen y si son machos se pelean entre ellos. Esos serán los métodos que definamos en la clase, además del constructor, métodos get/set y toString().

Si un gato maulla, mostrará por consola “Miauuu”, si un gato ronronea mostrará por consola “mrrrr”, etc.. Para hacer que un gato coma, hay que darle pescado con lo que el gato te dará las gracias mostrando el correspondiente texto por consola, cualquier otra comida la rechazará. Si dos gatos se pelean o no indicar la acción por consola.

NOTA: Usa el método *equals* de la clase *String* siempre que necesites comprobar el valor de una cadena de caracteres.

Crear un programa Principal, que cree una instancia para los gatos: **garfield** que come pescado, **tom** que come sopa y **lisa** que come carne. Todos los gatos maúllan y ronroneam. Además, garfield peleará con tom.

1. **FIZZ BUZZ**

Escribe un programa que muestre por consola los números de 1 a 100 (ambos incluidos), sustituyendo los siguientes:

* Múltiplos de 3 por la palabra "fizz".
* Múltiplos de 5 por la palabra "buzz".
* Múltiplos de 3 y de 5 a la vez por la palabra "fizzbuzz".

Crea un método que te diga si un número es divisible por 3, y otro que te diga si divisible por 5.

1. **Coche**

Diseñar una clase llamada *Coche* que tenga los atributos de clase (estáticos) *cochesCreados* y *kilometrosTotales*, así como el atributo de instancia *kilometrosRecorridos*.

Los coches, recorren una distancia determinada. Cuando un coche recorre una distancia, se incrementan su propio cuentakilómetros, es decir, su atributo kilometrosRecorridos y también se incrementa la cuenta global de kilómetros que recorren todos los coches, es decir, el atributo de clase *kilometrosTotales*.

El atributo *cochesCreados* se incrementa cada vez que se crea un coche.

La clase *Coche* además, tendrá un constructor, métodos get/set (solo los necesarios) y toString().

Crea un programa *Principal*, que cree una instancia de tres coches, que cada uno recorra un número de km aleatorios. Muestra al final, el número de km que ha recorrido cada coche, así como los km totales y número de coches totales creados.

1. **Cubo**

Diseñar una clase llamada *Cubo*. Todos los cubos tienen una determinada capacidad, un color, están hechos de un determinado material (que puede ser plástico o latón), y puede que tengan asa o puede que no. Un cubo se fabrica con el propósito de contener líquido; por tanto, otra característica es la cantidad de litros de líquido que contiene en un momento determinado. Nos interesa saber la capacidad máxima y los litros que contiene el cubo en cada momento, así que esos serán atributos que tendremos en cuenta.

Hay que tener en cuenta, que el cubo se puede vaciar o llenar por completo. También se puede volcar el contenido de un cubo en otro, recuerda antes de echar el líquido en otro cubo, que has de comprobar cuánto le cabe al cubo destino, teniendo en cuenta su capacidad máxima y el líquido que ya contiene, y volcar solo el líquido que se pueda, si se puede.

La clase Cubo además, tendrá un constructor, métodos get/set (solo los necesarios) y toString().

Crea un programa *Principal*, donde crees al menos dos cubos, y pasa líquido de uno a otro. Observa lo que se muestra por pantalla.

1. **Fracción**

Crea la clase *Fracción*. Los atributos serán: *signo*, *numerador* y *denominador*. Implementa, los métodos: invierte y toString().

A la hora de crear una fracción, ten en cuenta que una fracción no puede tener como denominador el número 0, así como calcula cual es el signo en función del numerador y denominador dado en el constructor. Además, tanto el numerado como el denominador, se expresarán en valores absolutos (usa Math.abs).

El método invierte, devuelve una nueva fracción invertida. Lo que antes era el numerador ahora será el denominador y viceversa.

El método toString, tal que represente la fracción en la forma: numerador / denominador y se muestre el signo “-“ delante en el caso que la fracción sea negativa.

1. **Pizza**

Crea la clase *Pizza* con los atributos y métodos necesarios. Sobre cada pizza, se necesita saber el tamaño - mediana o familiar - el tipo - margarita, cuatro quesos o funghi - y su estado - pedida o servida. La clase debe almacenar información sobre el número total de pizzas que se han pedido y que se han servido. Siempre que se crea una pizza nueva, su estado es “pedida”. A continuación, se muestra, un ejemplo de salida que puede dar la salida del programa:

pizza margarita mediana, pedida

pizza funghi familiar, servida

pizza cuatro quesos mediana, pedida

esa pizza ya se ha servido

pedidas: 3

servidas: 1

1. **Tiempo**

Crea la clase *Tiempo* con los métodos **suma** y **resta**. Los objetos de la clase Tiempo son intervalos de tiempo y se crean de la forma Tiempo t = new Tiempo(1, 20, 30), donde los parámetros que se le pasan al **constructor** son las horas, los minutos y los segundos respectivamente. Crea el método **toString** para ver los intervalos de tiempo de la forma 10h 35m 5s. Si se suman por ejemplo 30m 40s y 35m 20s el resultado debería ser 1h 6m 0s.

Realiza un programa de prueba para comprobar que la clase funciona bien. Crea al menos tres instancias y realiza entre ellos las operaciones de suma y de resta. Usa el método toString() para mostrar los intervalos de tiempo.

Nota: Internamente, la clase solo almacena los segundos totales. A partir de éstos, se puede deducir todo lo demás.

Ejemplo:

Tiempo t1 = **new** Tiempo(1, 20, 30);

Tiempo t2 = **new** Tiempo(0, 30, 40);

Tiempo t3 = **new** Tiempo(0, 35, 20);

System.out.println(t1 + " + " + t2 + " = " + t1.suma(t2));

System.out.println(t2 + " - " + t3 + " = " + t2.resta(t3));

1. **Venta de entradas**

Queremos gestionar la venta de entradas (no numeradas) de Fenavin que tiene 3 zonas, la sala principal con 1000 entradas disponibles, la zona de compra-venta con 200 entradas disponibles y la zona vip con 25 entradas disponibles. Hay que controlar que existen entradas antes de venderlas.

Crea una clase Zona donde se guarden las entradas disponibles, con su constructor correspondiente y un método que muestre las entradas que quedan por vender.

Además, deberá existir un método que venda un número de entradas. Este método debe comprobar si quedan entradas libres antes de realizar la venta. En el caso que no queden entradas libres, o bien las que hay no son suficientes, debe indicarlo con un mensaje por consola. Por ejemplo: “Lo siento, no quedan entradas libres”, o “Solo quedan XXX entradas para esta zona”. Si por contrario, puede realizarse la venta, mostrará el siguiente mensaje: “Aquí tiene sus XXXX entradas, gracias”.

El programa contará con un menú con las siguientes opciones: 1. Mostrar número de entradas libres 2. Vender entradas y 3. Salir

Si se elige la opción 1, deberán mostrarse las entradas libres, indicando la zona correspondiente. Si se elige la opción 2, se debe preguntar para qué zona queremos las entradas y cuántas queremos. Lógicamente, el programa debe controlar que no se puedan vender más entradas de la cuenta.

1. **Terminal Movil**

Implementa la clase Movil. Un móvil tiene asociado un número de teléfono y el tiempo de conversación máximo en segundos. Los móviles se pueden llamar unos a otros indicando el tiempo que van a estar conversando en segundos, tal que el tiempo de conversación corre para ambos. Además, cada móvil lleva asociada una tarifa que puede ser “rata”, “mono” o “bisonte”. El coste por minuto es de 6, 12 y 30 céntimos respectivamente. Se tarifican los segundos exactos. Obviamente, cuando un móvil llama a otro, se le cobra al que llama, no al que recibe la llamada.

Realiza un programa de prueba para comprobar que la clase funciona bien. Crea al menos un móvil para cada tarifa, que se llamen entre ellos y finalmente utilice el método toString para mostrar la información sobre cada móvil.

Si se llama al método toString, debe mostrar la información como se muestra en el siguiente ejemplo:

Nº 678 11 22 33 - 0s de conversación - tarificados 0,00 euros

1. **Ameba**

Las amebas son seres unicelulares de forma cambiante ya que carecen de pared celular. Fagocitan cualquier cosa que se les pone por delante.

Crea la clase Ameba con el atributo peso, el cual es un número entero que indica los microgramos que pesa el bicho. Al tratarse de una unidad tan pequeña, no se tienen en cuenta los decimales, será un dato entero. Cuando Dios crea una ameba de la nada – *new Ameba()* – su peso es de 3 microgramos. Al comer, va incrementando su peso; gasta un microgramo en el proceso de fagocitar y el resto hace que aumente de peso.

Por ejemplo, si come una partícula de 6 microgramos – por ej. miAmeba.come(6) – engordaría 5 microgramos.

Una ameba se puede comer a otra ameba. En este caso, sucede lo mismo que anteriormente, se gasta un microgramo en el proceso de fagocitado y el resto lo engorda la ameba que come.

Por ejemplo, si una ameba de 7 microgramos se come a una de 4, acaba pesando 10 microgramos. La ameba comida no se destruye, sino que se quedaría con un peso de 0 microgramos, una pena de ameba vamos.

Posteriormente, una ameba comida podría recuperarse si ella misma come algo.

La ameba tendrá un método toString() que mostrará cuanto pesa, ejemplo: “Soy una ameba y peso xxx microgramos”.

Implementa el método comer sobrecargándolo, tendrás un método para comer “comida” y otro que coma a otra ameba.

Realiza un programa de prueba, que cree al menos 2 amebas, que primero coman por si solas, luego una se coma a la otra y para terminar, vuelvan a comer por sí solas. Se debe ir llamando al método toString(), para saber cuánto pesa cada ameba tras cada acción realizada.

1. **Tarjeta Regalo**

La empresa El Corte Islandés nos ha encargado una aplicación para gestionar las tarjetas regalo.

Como primer paso para implementar la aplicación, es necesario crear la clase principal. Implementa la clase **TarjetaRegalo**. Cuando se crea una nueva tarjeta, se le da un saldo y se asigna de forma automática un número de 5 dígitos. Si se intenta **gastar** más dinero del que tiene la tarjeta, se debe mostrar un mensaje de error, por ejemplo “No tiene suficiente saldo para gastar 200.00€”.

Dos tarjetas regalo se pueden **fusionar**, creando una nueva tarjeta con la suma del saldo que tenga cada una y un nuevo número aleatorio de 5 cifras. Al fusionar dos tarjetas en una, las dos tarjetas originales se quedarían con 0 € de saldo.

Realiza un programa de prueba, que cree al menos 2 tarjetas regalo, que en primer lugar gasten dinero y finalmente se fusionen.

Usa el método toString() para mostrar la información de las tarjetas. Ejemplo: Tarjeta nº 67324 – Saldo 100.00€

1. **Calculadora**

Se desea tener una clase llamada **Calculadora**. Se debe crear en un paquete llamado **es.maestre**.

De esta clase NO se van a crear objetos, solo va a tener métodos de utilidad estáticos ( **static** )

Se pide implementar los siguientes métodos:

* **potencia**, que dada la base y el exponentes, nos devuelve el resultado de elevar la base al exponente.

public static int potencia(int base, int expo)

* **factorial**, que dado un número entero, nos devuelve el factorial de dicho número de forma recursiva

public static long factorial(int numero)

* **esPrimo**, que dado un número entero, nos devuelve true si es primo o false si no lo es.

Realiza un programa de prueba, que llame a cada uno de los métodos de la clase Calculadora.

1. **Cuenta corriente**

Implementa la clase *CuentaCorriente*. Cada cuenta corriente tiene un número de cuenta de 10 dígitos. Para simplificar, el número de cuenta se genera de forma aleatoria cuando se crea una cuenta nueva. La cuenta se puede crear con un saldo inicial; en caso de no especificar saldo, se pondrá a cero inicialmente.

En una cuenta se pueden hacer ingresos y gastos. También es posible hacer una transferencia entre una cuenta y otra. Se permite el saldo negativo.

Realiza un programa de prueba, que cree 3 cuentas corrientes, una de ellas inicialmente sin dinero. Realiza ingresos y gastos, así como transferencias entre ellas.

Usa el método toString() para mostrar la información de las cuentas corrientes.

Ejemplo: *Número de cc: 9319536518 Saldo: 1500,00 €*

1. **Ficha Dominó**

Implementa la clase *FichaDomino*. Una ficha de dominó tiene dos lados y en cada lado hay un número del 1 al 6 o bien ningún número (blanco).

Cuando se crea una ficha, se pueden proporcionar ambos valores, o bien se pueden querer crear de manera aleatoria usando el constructor por defecto.

Dos fichas **encajan** si se pueden colocar una al lado de la otra según el juego del dominó, por ejemplo, las fichas [ 2 | 5 ] y [ 4 | 5 ] encajan porque se pueden colocar de la forma [ 2 | 5 ] [ 5 | 4]. El método encajar devolverá true/false en función de si las fichas encajan o no. En el caso de encajar, mostrarán por consola la forma en la que encajan. Ejemplo: [6|1] [1|6]

Las fichas puedes **voltearlas**, tal que la ficha [ 2 | 5 ] se convierta en la [ 5 | 2]. El método voltear, solo puede ser usado internamente por el método encajar, si así lo necesitara, para poder encajar las fichas.

Realiza un programa de prueba, que cree al menos 3 fichas. Prueba a encajarlas.

Usa el método toString() para mostrar la información de las fichas.

Ejemplo: [6|1]

1. **Control de Incidencias**

Una empresa quiere implementar un programa que lleve el control de las incidencias que se producen en sus ordenadores.

Cada incidencia tiene un código: 1, 2, 3, 4, etc. Cuando se crea una nueva incidencia, se le asigna un código de forma automática y se pone el estado como “pendiente”.

Al crear una incidencia hay que indicar también el número de puesto (un número

entero).

Ejemplo: Incidencia inc1 = **new** Incidencia(105, "No tiene acceso a internet");

Cuando se **resuelve** una incidencia, hay que proporcionar información sobre cómo se ha resuelto o qué es lo que fallaba, además, el estado pasa a “resuelta”.

Ejemplo: inc1.resuelve("El equipo no tenía configurada la puerta de enlace");

Se deberá llevar la cuenta de las incidencias en estado “Pendiente”. Crea un método estático que te devuelva el número de incidencias pendientes.

Realiza un programa de prueba, que cree al menos 4 incidencias. Resuelve alguna de ellas.

Usa el método toString() para mostrar la información de las incidencias.

Ejemplo: Incidencia 1 - Puesto: 105 - No tiene acceso a internet - Pendiente

Finaliza el programa de prueba, mostrando el número de incidencias pendientes de resolver. Ejemplo: Incidencias pendientes: 3

1. **Batalla Pokemon**

Crea una clase Batalla con un método que calcule el daño durante una batalla entre dos Pokemon.

* La fórmula será la siguiente: daño = 50 \* (ataque / defensa) \* efectividad
* Efectividad: x2 (súper efectivo), x1 (neutral), x0.5 (no es muy efectivo)
* Sólo hay 4 tipos de Pokémon: Agua, Fuego, Planta y Eléctrico
* La batalla recibe los siguientes parámetros:
  + Tipo del Pokémon atacante.
  + Tipo del Pokémon defensor.
  + Ataque del Pokémon atacante: Valor aleatorio entre 1 y 100.
  + Defensa del Pokémon defensor: Valor aleatorio entre 1 y 100.

Crea un método para calcular la efectividad en función del Pokémon atacante y defensor:

* La efectividad por defecto es neutral
* Si el Pokémon atacante es igual al defensor: no es muy efectivo
* Si el Pokémon atacante es Agua y el defensor es Fuego o Eléctrico, o bien el Pokémon atacante es Planta o Fuego y el defensor es Agua: super efectivo

NOTA: Usa Enumerados para el tipo de Pokémon y la efectividad

Realiza un programa que pruebe la totalidad de la funcionalidad implementada.