**Práctica 1: Conversión**

La gravedad de la Luna es aproximadamente el 17% de la gravedad de la Tierra. Se pide:

1. Crear un programa que, dado un peso de la Tierra, calcule su equivalente en la Luna.
2. El programa debe mostrar por pantalla un mensaje similar a este: “Un peso de 30 kilos en la Tierra equivalen a X kilos en la Luna”.

**Práctica 2: Operaciones matemáticas**

Crear un programa que realice algunas operaciones matemáticas.

1. Pedir por consola al usuario que introduzca 3 números enteros y almacenar su valor en variables.
2. Operación 1. a*b/c Mostrar por consola el resultado
3. Operación 2. (a*c)%bMostrar por consola el resultado
4. Operación 3. 2*(a+c-b)/(b*c) Mostrar por consola el resultado
5. Operación 4. ((a*c)+(b%a))/a-c Mostrar por consola el resultado
6. Opcional: Repetir las operaciones con números decimales para ver las diferencias de resultado.

**Práctica 3: Notas**

Crear un programa que indique al usuario la nota media global de su curso. Requisitos:

1. El programa pedirá al usuario que introduzca 5 notas, para las asignaturas: Matemáticas, Física, Química, Lenguaje e Historia.
2. El programa realizará la media de esas cinco notas.
3. El programa indicará al usuario el rango al que equivale la nota final que ha sacado:
    a. Entre 0 – 3 Muy deficiente
    b. Entre 3 – 5 Insuficiente
    c. Entre 5 – 6 Suficiente
    d. Entre 6 – 7 Bien
    e. Entre 7 – 9 Notable
    f. Entre 9 – 10 Sobresaliente

**Práctica 4: Contar espacios**

Crear un programa que lea caracteres desde teclado indefinidamente. Requisitos:

1. El programa pedirá al usuario que inserte caracteres o frases completas por teclado de manera indefinida
2. El programa contará el número de espacios que se van introduciendo por consola
3. El programa finalizará cuando el usuario introduzca un punto “.”
4. El programa imprimirá por consola el número total de espacios que se han introducido

**Práctica 5: Amstrong**

Desarrollar un programa que determine si un número es un número de Amstrong. Un número de Amstrong es aquel que es igual a la suma de sus dígitos elevados a la potencia de su número de cifras.

Recursos para el ejercicio:

- Math.floor: Devuelve el máximo entero menor o igual a un número pasado como parámetro.
- Math.log10: Devuelve el logaritmo en base 10 de un número pasado como parámetro.
- Math.pow: Devuelve el valor del primer argumento elevado a la potencia del segundo argumento.

**Práctica 6: Conversión 2**

Al igual que se hizo con la Práctica 1 de conversión de pesos, en este caso se pide crear un programa que imprima una tabla de conversión de pulgadas a metros. Requisitos:

1. Un metros son 39,37 pulgadas
2. Elaborar el programa desde la pulgada 1 hasta la 144
3. La tabla debe de dejar un espacio libre cada 12 pulgadas para ser más legible.

**Práctica 7: CandyCalculator**

En las fiestas de un barrio hay competiciones deportivas que premian el desempeño en ellas con cupones que luego puedes cambiar por golosinas. Una barra de caramelo se puede cambiar por 10 cupones, y un chicle por 3 cupones.

Escribe una clase CandyCalculator que tenga un método candyCalculator que permita:

1. Calcular, dado un número dado de cupones, cuantas barras de caramelo y chicles puedes obtener si gastas todos tus cupones en barras de caramelo primero, y utilizas los cupones restantes en chicles.
2. También te tiene que devolver el número de cupones restantes que no puedes gastar.
3. Devuelve los valores en un array en donde la posición [0] representa las barras de caramelo, la posición [1] los chicles y la posición [2] los cupones restantes.

**Práctica 8: Elementos Duplicados**

Crear un programa que, dado un array de números enteros, determine cuales son sus elementos que se encuentran duplicados. Por ejemplo:

int [] arrayDePrueba = {1, 2, 3, 3, 9, 8, 7, 4, 6, 7, 0, 4, 5};

**Práctica 9: Contar vocales**

Implementa la función int contarVocales(String) que dada una cadena, cuenta el número de vocales que existe en la misma. No importa que las vocales estén en mayúsculas o minúsculas y el resto de caracteres que no sean vocales son ignorados. También se ignoran las vocales acentuadas.

**Práctica 10: Persona**

Se requiere un programa que modele el concepto de una persona. Una persona posee nombre, apellido, número de documento de identidad y año de nacimiento. La clase debe tener un constructor que inicialice los valores de sus respectivos atributos.

La clase debe incluir los siguientes métodos:

1. Definir un método que imprima por pantalla los valores de los atributos del objeto.
2. En el método main se deben crear dos personas y mostrar los valores de sus atributos por pantalla.

**Práctica 11: Rectángulo**

Crea una clase Rectángulo con las siguientes características

- Incluyen un constructor al que se le pasan los datos de base y altura. Si se intenta dar valor negativo a alguna de las dimensiones, lo corrige al valor positivo usando Math.abs(int a).
- Un constructor sin parámetros que inicializa un nuevo rectángulo con base 2 y altura 1.
- Un método esCuadrado que indica si el rectángulo actual es un cuadrado.
- Un método area que calcula el área del rectángulo actual.
- Un método perimetro que calcula el perímetro del rectángulo actual.
- Un método gira que gira 90 grados el rectángulo actual (intercambiado la base por la altura).

**Práctica 12: Bag**

Crea una clase Bag que nos permita almacenar una colección desordenada de objetos en la que, al extraerlos, todos tengan la misma probabilidad de ser escogidos.

En el archivo adjunto tenéis el código con los métodos que hay que crear y una explicación de cómo deben funcionar.

**Práctica 13: Cuenta Bancaria**

Desarrollar un programa que modele una cuenta bancaria que tiene los siguientes atributos, que deben ser de acceso protegido:

- Saldo, de tipo float.
- Número de ingresos con valor inicial cero, de tipo int.
- Número de retiros con valor inicial cero, de tipo int.
- Tasa anual (porcentaje), de tipo float.
- Comisión mensual con valor inicial cero, de tipo float.

La clase Cuenta tiene un constructor que inicializa los atributos saldo y tasa anual con valores pasados como parámetros. La clase Cuenta tiene los siguientes métodos:

- Ingresar una cantidad de dinero en la cuenta actualizando su saldo.
- Retirar una cantidad de dinero en la cuenta actualizando su saldo. El valor a retirar no debe superar el saldo.
- Calcular el interés mensual de la cuenta y actualiza el saldo correspondiente.

- Extracto mensual: actualiza el saldo restándole la comisión mensual y calculando el interés mensual correspondiente (invoca el método anterior).
- Imprimir: muestra en pantalla los valores de los atributos.

La clase Cuenta tiene dos clases hijas:

- Cuenta de ahorros: posee un atributo para determinar si la cuenta de ahorros está activa (tipo boolean). Si el saldo es menor a 100€, la cuenta está inactiva, en caso contrario se considera activa. Los siguientes métodos se redefinen:
    1. Ingresar: se puede ingresar dinero si la cuenta está activa. Debe invocar al método heredado.
    2. Retirar: es posible retirar dinero si la cuenta está activa. Debe invocar al método heredado.
    3. Extracto mensual: si el número de retiros es mayor que 4, por cada retiro adicional, se cobra 1.5€ como comisión mensual. Al generar el extracto, se determina si la cuenta está activa o no con el saldo.
    4. Un nuevo método imprimir que muestra en pantalla el saldo de la cuenta, la comisión mensual y el número de transacciones realizadas (suma de cantidad de consignaciones y retiros).
- Cuenta corriente: posee un atributo de sobregiro, el cual se inicializa en cero. Se redefinen los siguientes métodos:
- Retirar: se retira dinero de la cuenta actualizando su saldo. Se puede retirar dinero superior al saldo. El dinero que se debe queda como sobregiro.
- Ingresar: invoca al método heredado. Si hay sobregiro, la cantidad consignada reduce el sobregiro.
- Extracto mensual: invoca al método heredado.
- Un nuevo método imprimir que muestra en pantalla el saldo de la cuenta, la comisión mensual, el número de transacciones realizadas (suma de cantidad de consignaciones y retiros) y el valor de sobregiro.

Realizar un método main que implemente un objeto Cuenta de ahorros y llame a los métodos correspondientes

**Práctica 14: Universidad**

En la universidad existe una plantilla de personal que está dividida en dos grupos:

- Personal de Administración y Servicios (PAS): En el personal de administración se encuentra todo el personal que desempeña labores administrativas o de gestión de algún tipo de servicio. En este caso, por simplicidad, solo vamos a considerar que existe personal administrativo y personal informático.
- Personal Docente Investigador (PDI): Personal que se dedica a la docencia y a la investigación y en el que incluimos a profesores y a investigadores contratados en algún proyecto de investigación.

Todo el personal se identifica con su nombre y DNI y recibe un salario mensual que debemos de calcular, de forma simplificada, como el número de horas de trabajo a la semana multiplicado por el importe/hora de su categoría y por el número de semanas al mes (4).

Los datos de cada tipo de personal son los siguientes (las cantidades no son reales):

- Administrativos: 37 horas/semana y un importe de 7,5 euros/hora.
- Informáticos: 40 horas/semanas y un importe de 6 euros/hora.
- Profesores: 37 horas/semana con un importe de 8 euros/hora.
- Investigadores: 35 horas/semana con un importe de 7 euros/hora.

Además los profesores y los PAS tienen los siguientes ingresos adicionales:

- Profesores: A los profesores se les añade un complemento salarial en función de lo que se conoce como sexenios (periodos de 6 años evaluados positivamente), y que se pueden conceder hasta un máximo de 6. En concreto, se les añade 100 euros al mes por sexenio reconocido. Una vez concedido el sexenio el aumento de sueldo se mantiene para siempre.
- PAS: En un mes determinado los PAS (tanto administrativos como informáticos) pueden acumular horas extras. Cada hora extra se paga a 6 euros/hora. Las horas extra, una vez pagadas, se inicializan a cero de nuevo (por ejemplo, a la hora de calcular el sueldo de personal).

Se pide:

- Crear una estructura de clases y subclases que represente a los trabajadores de la universidad. En dicha estructura tiene que existir una superclase común a todos los trabajadores que sea Personal.
- Crear una clase Universidad con una función public static void imprimirNominas(Personal[] listaPersonal) que imprima por pantalla, para cada Personal incluido en listaPersonal, el nombre del personal, su categoría y su sueldo.
- Crear en la misma clase Universidad, una función obtenerPresupuestoTotal que devuelva el total de dinero que se gasta la universidad en personal.

Para evitar usar decimales en cantidades monetarias podemos trabajar con el dinero en céntimos de Euro y solo convertirlo en euros a la hora de mostrarlo por pantalla.

**Práctica 15: Cartas**

Modifica la clase Mazo de las cartas para que pueda ordenarse según distintos criterios y algoritmos de ordenación. Para ello:

1. Crea una clase abstracta AlgoritmoOrdenacion que incluya un método abstracto ordenar(List listaCartas) que devuelve la lista de cartas pasadas por parámetro ordenadas.
2. Crea subclases de AlgorimoOrdenacion que ordenan las cartas por diferentes criterios:
   - Ordenación palo-número-incremental: Ordena las cartas primero por el palo (por orden alfabético de BASTOS, COPAS, ESPADAS, OROS) y luego por el número (de más pequeño a más grande).
   - Ordenación palo-número-decremental: Ordena las cartas primero por el palo (por orden alfabético de BASTOS, COPAS, ESPADAS, OROS) y luego por el número (de más grande a más pequeño).
   - Ordenación número-palo. Ordena a las cartas primero por número (incremental) y luego por palo (orden alfabético)

- Ordenación palo-número-incremental: Ordena las cartas primero por el palo (por orden alfabético de BASTOS, COPAS, ESPADAS, OROS) y luego por el número (de más pequeño a más grande).
- Ordenación palo-número-decremental: Ordena las cartas primero por el palo (por orden alfabético de BASTOS, COPAS, ESPADAS, OROS) y luego por el número (de más grande a más pequeño).
- Ordenación número-palo. Ordena a las cartas primero por número (incremental) y luego por palo (orden alfabético)
    1. Puedes buscar algoritmos de ordenación en: https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento y necesitarás especificar el criterio de ordenación de cartas.
    2. Permite que se le pueda indicar el algoritmo de ordenación al mazo mediante un método setAlgoritmo
    3. Ordena el mazo mediante el algoritmo de ordenación que le has indicado, el mazo no sabe realmente cómo se está ordenando En el ejemplo ya tenéis creado la mayoría del código de las clases, solo hay que introducir el código correspondiente a la ordenación. Es decir, los métodos setAlgoritmo y ordena de Mazo y las clases OrdPalNumDec, OrdPalNumInc y OrdNumIncPal.

En cada método de ordenación os será útil crear un comparador de cartas usado en ese algoritmo En el main se muestra cómo se crea un mazo y se ordena de varias formas distintas.