APARTADO1 - CLASES POO

APARTADO1.Ejercicio1

Crea las clases Animal, Mamífero, Ave, Gato, Perro, Canario, Pingüino y Lagarto. Crea, al menos, tres métodos específicos de cada clase y redefine el/los métodos/s cuando sea necesario. Prueba las clases creadas en un programa en el que se instancien objetos y se les apliquen métodos.

APARTADO1.Ejercicio2

Crea una clase llamada Factura que una ferretería podría usar para representar una factura de un artículo vendido en la tienda. Una factura debe incluir cuatro datos como variables de instancia:

1. un número de pieza (tipo String),
2. una descripción de la pieza (tipo String)
3. una cantidad del artículo comprado (tipo int)
4. un precio por artículo (doble).

Su clase debe tener un constructor que inicialice las cuatro variables de instancia. Proporciona los mecanismos de modificación y obtención para cada atributo del objeto. Además, proporcione un método llamado obtenerFactura que calcula el importe de la factura (es decir, multiplica la cantidad por el precio por artículo), y luego devuelve el importe como un valor doble. Si la cantidad no es positiva, debe establecerse en 0. Si el precio por artículo no es positivo, debe establecerse en 0.0.

Escriba una aplicación de prueba llamada InvoiceTest que demuestre las capacidades de la clase Invoice.

APARTADO1.Ejercicio3

Crear una clase llamada Empleado que incluya tres piezas de información como variables de instancia:

1. un nombre (tipoString)
2. un apellido (tipoString)
3. un salario mensual (doble).

Tu clase debe tener un constructor que inicialice las tres variables de instancia. Proporcione un conjunto y un método de obtención para cada atributo de instancia. Si el salario mensual no es positivo, póngalo en 0.0.

Además, codifique un método que calcule el incremento en el sueldo del empleado llamado "incrementarSueldo(int porcentajeIncremento)" donde se le pasa el % que de incremento salarial aplicado al empleado. Escriba una aplicación de prueba llamada EmployeeTest que demuestre las capacidades de la clase Employee. Crea dos objetos Employee y muestra el salario anual de cada objeto. Luego, dale a cada Empleado un aumento del 10% y vuelve a mostrar el salario anual de cada Empleado.

APARTADO1.Ejercicio4

Crear una clase llamada "Libro" para representar un libro. Un Libro debe incluir cuatro piezas de información como atributos de instancia - un nombre de libro, un número ISBN, un nombre de autor y un editor. Tu clase debe tener un constructor que inicialice las cuatro variables de instancia. Proporcionar un método de modificación y acceso para cada atributo de instancia. Además, proporciona un método privado llamado getBookInfo que devuelve la descripción del libro como una cadena (la descripción debe incluir toda la información sobre el libro).

Se utilizará este método internamente tanto en los métodos de modificación como en el constructor para mostrar por terminal la descripción del objeto en ese momento.

Finalmente, escribir una aplicación de prueba llamada BookTest para crear un vector de objetos de 30 elementos para la clase Book para demostrar las capacidades de la clase Book.

APARTADO1.Ejercicio5

Crear la clase "CuentaAhorros". Utilice una variable estática pública anualInterestRate para almacenar la tasa de interés anual para todos los titulares de la cuenta. Cada objeto de la clase contiene un atributo de instancia balancePrivado que indica el monto que el ahorrador tiene actualmente en depósito y la cuál se debe inicializar mediante el constructor siempre que se crea un objeto de dicha clase.

Proporcione el método calcularInteresMensual para calcular el interés mensual multiplicando el balancePrivado por el anualInterestRate dividido por 12.

Proporcione un método estático modificarInterestRate que establece el Interés anual a un nuevo valor.

Escriba un programa para probar la clase "CuentaAhorros". Instanciar dos objetos de la Cuenta de Ahorros, Usuario1 y Usuario2, con saldos de 2000.00 y 3000.00, respectivamente. Establezca la tasa de interés anual al 4%, luego calcule el interés mensual e imprima los nuevos saldos para ambos ahorradores. Luego establezca la tasa de interés anual al 5%, calcule el interés del mes siguiente e imprima los nuevos saldos para ambos ahorradores.

APARTADO1.Ejercicio6

Crea la clase Fracción. Los atributos serán numerador y denominador. Y algunos de los métodos pueden ser invierte, simplifica, multiplica, divide, etc

APARTADO1.Ejercicio7

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". El siguiente código del programa principal debe dar la salida que se muestra:

```java
public class PedidosPizza {
  public static void main(String[] args) {
    Pizza p1 = new Pizza("margarita", "mediana");
    Pizza p2 = new Pizza("funghi", "familiar");
    p2.sirve();
    Pizza p3 = new Pizza("cuatro quesos", "mediana");
    System.out.println(p1);
    System.out.println(p2);
    System.out.println(p3);
    p2.sirve();
    System.out.println("pedidas: " + Pizza.getTotalPedidas());
    System.out.println("servidas: " + Pizza.getTotalServidas());
  }
}
```

```
pizza margarita mediana, pedida
pizza funghi familiar, servida
pizza cuatro quesos mediana, pedida
esa pizza ya se ha servido
pedidas: 3
servidas: 1
```

APARTADO1.Ejercicio8

Queremos gestionar la venta de entradas (no numeradas) de Expocoches "Campanillas" 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, para ello es necesario codificar un método "vender(int n)" el cuál permitirá realizar la venta siempre y cuando el número de entradas disponibles sea suficiente, en caso de no ser suficiente retorna un mensaje de error y no se realiza la venta solicitada.
En el programa principal se instanciarán 3 objetos por cada sala y se testeará.

Además, realizar un menú del programa debe ser el que se muestra a continuación.

- La opción1, muestra todas las entradas disponibles por cada zona.
- La opcion2, nos preguntará para qué zona queremos las entradas y cuántas queremos. Lógicamente, el programa debe controlar que no se puedan vender más entradas de la cuenta.
- La opcion3, termina el programa.

```
1. Mostrar número de entradas libres
2. Vender entradas
3. Salir
```

APARTADO1.Ejercicio9

Crea una clase de funciones matemáticas llamada “Matematicas” que contenga los siguientes métodos. Recuerda que puedes usar unas dentro de otras si es necesario.

1. esCapicua: Devuelve verdadero si el número que se pasa como parámetro es capicúa y falso en caso contrario.
2. esPrimo: Devuelve verdadero si el número que se pasa como parámetro es primo y falso en caso contrario.
3. siguientePrimo: Devuelve el menor primo que es mayor al número que se pasa como parámetro.
4. potencia: Dada una base y un exponente devuelve la potencia.
5. digitos: Cuenta el número de dígitos de un número entero.
6. voltea: Le da la vuelta a un número.
7. digitoN: Devuelve el dígito que está en la posición n de un número entero. Se empieza contando por el 0 y de izquierda a derecha.
8. posicionDeDigito: Da la posición de la primera ocurrencia de un dígito dentro de un número entero. Si no se encuentra, devuelve -1.
9. quitaPorDetras: Le quita a un número n dígitos por detrás (por la derecha).
10. quitaPorDelante: Le quita a un número n dígitos por delante (por la izquierda).
11. pegaPorDetras: Añade un dígito a un número por detrás.
12. pegaPorDelante: Añade un dígito a un número por delante.
13. trozoDeNumero: Toma como parámetros las posiciones inicial y final dentro de un número y devuelve el trozo correspondiente.
14. juntaNumeros: Pega dos números para formar uno

APARTADO1.Ejercicio10

Crea una clase que contenga funciones de manejo de vectores (arrays de una dimensión) de números enteros que contenga las siguientes funciones.

- generaArrayInt: Genera un array de tamaño n con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
- minimoArrayInt: Devuelve el mínimo del array que se pasa como parámetro.
- maximoArrayInt: Devuelve el máximo del array que se pasa como parámetro.
- mediaArrayInt: Devuelve la media del array que se pasa como pará- metro.
- estaEnArrayInt: Dice si un número está o no dentro de un array.
- posicionEnArray: Busca un número en un array y devuelve la posición (el índice) en la que se encuentra.
- volteaArrayInt: Le da la vuelta a un array.
- rotaDerechaArrayInt: Rota n posiciones a la derecha los números de un array.
- rotaIzquierdaArrayInt: Rota n posiciones a la izquierda los números de un array

Se utilizará como constructor el constructor por defecto sin inicializar la matriz.

APARTADO1.Ejercicio11

Crea una biblioteca de funciones de manejo de matrices (arrays de dos dimensiones) de números enteros que contenga las siguientes funciones.

- generaArrayBiInt: Genera un array de tamaño n x m con números aleatorios cuyo intervalo (mínimo y máximo) se indica como parámetro.
- filaDeArrayBiInt: Devuelve la fila i-ésima del array que se pasa como parámetro.
- columnaDeArrayBiInt: Devuelve la columna j-ésima del array que se pasa como parámetro.
- coordenadasEnArrayBiInt: Devuelve la fila y la columna (en un array (vector) con dos elementos) de la primera ocurrencia de un número dentro de un array bidimensional. Si el número no se encuentra en el array, la función devuelve el array {-1, -1}.
- esPuntoDeSilla: Dice si un número es o no punto de silla, es decir, mínimo en su fila y máximo en su columna.

Se utilizará como constructor el constructor por defecto sin inicializar la matriz.

APARTADO2. POO – ARRAYS DE OBJETOS.

APARTADO2.Ejercicio1

Utiliza la clase Gato para crear un array de cuatro gatos e introduce los datos de cada uno de ellos mediante un bucle. Muestra a continuación los datos de todos los gatos utilizando también un bucle

APARTADO2.Ejercicio2

Cambia el programa anterior de tal forma que los datos de los gatos se introduzcan directamente en el código de la forma gatito[2].setColor("marrón") o bien mediante el constructor, de la forma gatito[3] = new Gato("Garfield", "naranja", "macho"). Muestra a continuación los datos de todos los gatos utilizando un bucle.

APARTADO2.Ejercicio3

Primero se define la clase Disco. Con los siguientes atributos:

- String codigo: Será un número aleatorio entero y positivos. (Math.abs(new Random().nextInt()))
- String autor
- String titulo
- String genero
- int duración

A su vez tendrá los getters y setters de todos los atributos, salvo de "codigo" que solo dispondremos del get, siendo un valor no modificable por el usuario.

Crearemos una clase ColeccionDeDiscos donde inicializamos un vector de 10 discos. Esta clase tendrá también dos métodos, uno para añadir un disco dentro de la colección y otro para retirar un disco de la colección, a la hora de retirar un disco el usuario podrá escoger cuál retirar.

Finalmente crearemos un main que utilice la clase ColeccionDeDiscos y disponga un menú con las siguientes opciones:

- Listar discos en la colección de Discos.
- Añadir disco. Solicitamos al usuario todos los datos necesarios, es decir, autor, título, género y duración. Pues código es un atributo que crea Disco de forma aleatoria y según su propia lógica.
  IMPORTANTE: Se añadirá el Disco en la primera posición disponible dentro del vector de coleccionDiscos.
- Retirar disco. Solicitamos al usuario que indique que disco retiramos mediante el ID o codigo de referencia asignado al disco.

Ahora solicitaremos por pantalla al usuario que introduzca los datos de autor, título, genero y duración. Estos datos los pasaremos al constructor de la clase Disco, siendo el atributo "codigo" completo por la propia clase en función de su propia lógica de etiquetado.

APARTADO2.Ejercicio4

Modifica el programa "Colección de discos" como se indica a continuación:

a) Mejora la opción "Nuevo disco" de tal forma que cuando se llenen todas las posiciones del array, el programa muestre un mensaje de error. No se permitirá introducir los datos de ningún disco hasta que no se borre alguno de la lista.
b) Mejora la opción "Borrar" de tal forma que se verifique que el código introducido por el usuario existe.
c) Modifica el programa de tal forma que el código del disco sea único, es decir que no se pueda repetir.
d) Crea un submenú dentro dentro de "Listado" de tal forma que exista un listado completo, un listado por autor (todos los discos que ha publicado un determinado autor), un listado por género (todos los discos de un género determinado) y un listado de discos cuya duración esté en un rango determinado por el usuario.

APARTADO2.Ejercicio4.2: Colección de discos ampliado (con canciones)

Este ejercicio consiste en realizar una mejora del ejercicio anterior que permite gestionar una colección de discos. Ahora cada disco contiene canciones. Deberás crear la clase Cancion con los atributos y métodos que estimes oportunos. Añade el atributo canciones a la clase Disco. Este atributo canciones debe ser un array de objetos de la clase Cancion.
Fíjate bien que Cancion NO ES una subclase de Disco sino que ahora cada disco puede tener muchas canciones, que están almacenadas en el atributo canciones.
Modifica convenientemente el método toString para que al mostrarse los datos de un disco, se muestre también la información sobre las canciones que contiene. La aplicación debe permitir añadir, borrar y modificar las canciones de los discos.

APARTADO2.Ejercicio5

Crea el programa GESTISIMAL (GESTIón SIMplificada de Almacén) para llevar el control de los artículos de un almacén. En un almacén podrá haber como máximo 50 artículos diferentes. De cada artículo se debe saber el código, la descripción, el precio de compra, el precio de venta y el stock (número de unidades). El menú del programa debe tener, al menos, las siguientes opciones:

```
1. Listado
2. Alta
3. Baja
4. Modificación
5. Entrada de mercancía
6. Salida de mercancía
7. Salir
```

- Listar: Lista todos los artículos dando su estado actual.
- Alta: Dar de alta un producto con un número de stock inicial.
- Baja: Se retira un artículo completamente del almacén, pese que hubiese stock.
- Entrada: Seleccionado dado un código de artículo se añade X productos.
- Salida: Seleccionado dado un código de artículo su stock se reduce en X productos.

La entrada y salida de mercancía supone respectivamente el incremento y decremento de stock de un determinado artículo. Hay que controlar que no se pueda sacar más mercancía de la que hay en el almacén.

APARTADO2.Ejercicio6

Crea una clase llamada **Factura** que una ferretería podría usar para representar una factura de un artículo vendido en la tienda. Una factura va a estar conformada de uno o varios (max.10) objetos de la clase **Articulo** que debe incluir cuatro datos como variables de instancia:

5. un número de pieza (tipo String),
6. una descripción de la pieza (tipo String)
7. una cantidad del artículo comprado (tipo int)
8. un precio por artículo (doble).

La clase **Articulo** debe tener un constructor que inicialice las cuatro variables de instancia. Proporciona los mecanismos de modificación y obtención para cada atributo del objeto.

En la clase Factura debemos crear dos métodos uno **incluirArticulo(Articulo articulo)** donde agregamos un artículo a la factura. Y un método llamado **obtenerFactura()** que calcula el importe de la factura (es decir, multiplica la cantidad por el precio por artículo y por cada artículo incluido en la factura), y luego devuelve el importe como un valor doble.

Escriba una aplicación de prueba que demuestre las capacidades de la clase Factura donde ingresamos agregaremos diversos artículos y obtendremos la factura total.

APARTADO2.Ejercicio7

Construir una fábrica de coches. Para ello se deben codificar las siguientes clases.

1. **Clase Rueda**
    a. double radio
    b. 3 constantes estáticas tipo int:
        i. SECO = 0
        ii. HUMEDO = 1
        iii. NEVADO = 2
    c. int tipo: Se le asignará un tipo estático de los anteriores. Por defecto, SECO.
    d. **Rueda(double radio, String tipo)**
    e. **Rueda ():** Inicializa una rueda con el siguiente estado (radio = 1, tipo = SECO).
    f. Métodos **get/set y toString** (toString debe convertir el tipo en texto según la constante asignada actualmente a tipo).

Prueba:

```
Rueda ruedaHumedo = new Rueda(2, Rueda.HUMEDO);
Rueda ruedaDefault = new Rueda();

System.out.println(ruedaHumedo);
System.out.println(ruedaDefault);
```

```
Rueda [radio=2.0, tipo=HUMEDO]
Rueda [radio=1.0, tipo=SECO]
```

2. Clase Chasis.
    a. double peso
    b. 4 constantes estáticas tipo String:
        i. MATERIAL1 = “ALUMINIO”
        ii. MATERIAL2 = “ACERO”
        iii. MATERIAL3 = “HIERRO”
        iv. MATERIAL4 = “FIBRA”
    c. String material: Se le asignará un tipo estático de los anteriores. Por defecto, MATERIAL1.
    d. **Chasis (double peso, String material)**
    e. **Chasis ():** Inicializa un chasis con los siguientes valores por defecto (peso = 1000, material = MATERIAL1)
    f. Métodos **get/set y toString**

Prueba:

```
Chasis chasisCustom = new Chasis(2500, Chasis.MATERIAL2);
Chasis chasisDefault = new Chasis();

System.out.println(chasisCustom);
System.out.println(chasisDefault);
```

```
Chasis [peso=2500.0, material=ACERO]
Chasis [peso=1000.0, material=ALUMINIO]
```

3. Clase Coche
    a. 4 constantes estáticas tipo int:
        i. ROJO = 0
        ii. AZUL = 1
        iii. AMARILLO = 2
        iv. BLANCO = 3
    b. Rueda [] conjuntoRuedas: Arrays de 4 objetos Rueda
    c. Chasis chasis.
    d. int color: Se le asignará un tipo estático de los anteriores. Por defecto, BLANCO.
    e. **Coche (Rueda rueda, Chasis chasis, int color)**: Donde una copia de rueda se repetirá x4 para cada una de las ruedas del coche, inicializándose todas iguales.
    f. **toString()** . Imprime un String con la información del coche y sus componentes.
    g. **setColor(int color)** : modifica el color del coche.
    h. **setRueda(Rueda rueda, int pos)** : modifica una rueda en una posición determinada.
    i. **setChasis(Chasis chasis):** Modifica el chasis actual.

Prueba:

```
Coche coche1 = new Coche(ruedaHumedo, chasisCustom, Coche.AZUL);
System.out.println(coche1);
```

```
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO],
    Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]],
    chasis=Chasis [peso=2500.0, material=ACERO], color=1]
```

4. Clase Fabrica
    a. String nombreFabrica
    b. double defaultRadio
    c. int defaultTipo
    d. double defaultPeso
    e. String defaultMaterial
    f. Int capacidadFabrica: Número de coches que puede albergar una fábrica.
    g. Coche[] conjuntoCochesFabricados: Conjunto total de coches fabricados
    h. **Constructor** indicando todos los parámetros representativos de la fábrica (nombre, radio, tipo, peso, material y capacidad)
    i. Métodos **get/set** para los parámetros por defecto de la fábrica.
    j. Método público **iniciarFabricacion(int numeroCoches) : boolean**. Se inicia la fabricación de coches en caso de que se quiera fabricar más coches de los que actualmente puede albergar dará un error antes de iniciar su fabricación cancelando el proceso. Para la fabricación se utilizará el método privado "fabricarCoche()" en caso de fallo se mostrará un mensaje por pantalla y se intentará nuevamente. Cada coche nuevo debe ocupar un espacio en el array de "cochesFabricados".
    k. **Coche fabricarCoche():** Método privado que fabrica un coche. Devolviendo un objeto de la clase Coche con los atributos por defecto de la fábrica y el color del coche se escoge de forma aleatoria entre los disponibles (num. Entre 0 y 3 ambos incluídos).
    l. Método público **retirarCoche(int numeroCoches) : boolean**.
    Comprobación:
        - Primero es necesario comprobar si hay suficientes coches, sino lo hay se devolverá simplemente un false y no se hará nada.

    Actuación en caso de poder realizar la retirada:
        - Utilizar el método privado "sacarCoche" tantas veces como el número de coches a retirar y finalmente devolverá true.
        Mediante este método se retira siempre los coches por orden de fabricación el que está en primera posición del conjunto de cochesFabricados por cada coche a retirar – llamando a sacarCoche(0).

    m. Método público **retirarCoche(int numeroCoches, int color) : boolean.**
    Comprobaciones:
        - Si hay suficientes coches, sino lo hay se devolverá simplemente un false y no se hará nada.
        - Entre los coches disponibles hay suficientes coches del color indicado, sino lo hay se devolverá simplemente un false y no se hará nada.

    Actuación en caso de poder realizar la retirada:
        - Utilizar el método privado "sacarCoche" tantas veces como el número de coches a retirar y finalmente devolverá true.
        Mediante este método se retira siempre los coches por orden de fabricación el que está en las primeras posiciones y que tenga el color de coche deseado del conjunto de cochesFabricados por cada coche a retirar – llamando a sacarCoche(posicion). Siendo posición la posición del primer coche detectado del color deseado.

n. Método privado **sacarCoche(int posición)** : Retira el objeto Coche del conjunto de cochesFabricados situado en la posición indicada y actualiza el conjunto desplazando las posiciones de los coches en fabrica una posición menos desde la posición retirada.

TIP: Cada vez que retiremos un coche recoloca la colección de coches para que no queden huecos vacíos llevando los coches. Es decir, si tenemos [coche1, coche2, coche3, null] si quitamos el coche1 arrastramos coche2 a la posición de coche1 y coche3 a la posición de coche2. Resultado: [coche2, coche3, null,null].

o. **toString().** Imprime todos los coches fabricados hasta ese momento y disponibles en la fábrica y sus características.

Prueba:

```
Fabrica fabricaPepe = new Fabrica("PEPE", 2, Rueda.HUMEDO, 2500, Chasis.MATERIAL2, 10);
fabricaPepe.iniciarFabricacion(3);
System.out.println(fabricaPepe);

Fabrica fabricaJuan = new Fabrica("JUAN", 2, Rueda.NEVADO, 2000, Chasis.MATERIAL4, 5);
fabricaJuan.iniciarFabricacion(2);
System.out.println(fabricaJuan);
```

```
########################################
Fabrica: PEPE
Numero actual de fabricados: 3/10
******************************
Coche1 color AZUL:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]], chasis=Chasis [peso=2500.0, material=ACERO], color=AZUL]:
******************************
Coche2 color ROJO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]], chasis=Chasis [peso=2500.0, material=ACERO], color=ROJO]:
******************************
Coche3 color BLANCO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO], Rueda [radio=2.0, tipo=HUMEDO]], chasis=Chasis [peso=2500.0, material=ACERO], color=BLANCO]:

########################################
Fabrica: JUAN
Numero actual de fabricados: 2/5
******************************
Coche1 color BLANCO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO]], chasis=Chasis [peso=2000.0, material=FIBRA], color=BLANCO]:
******************************
Coche2 color ROJO:
Coche [conjuntoRuedas=[Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO], Rueda [radio=2.0, tipo=NEVADO]], chasis=Chasis [peso=2000.0, material=FIBRA], color=ROJO]:
```

Crea una clase auxiliar con el método main para testear las clases. Se debe instanciar dos fábricas con 100 y 50 de capacidad máxima de coches y habrá que testear tanto el proceso de iniciar nuevas fabricaciones como la retirada de los coches ya sea cogiendo coches independientemente del color como precisando el color deseado.

```
fabricaPepe.retirarCoche(10); // Retirar 10 cuyo color sea de aleatorio.
fabricaPepe.retirarCoche(10, Coche.AZUL); // Retirar 10 coches azules.
```

PASOS (Entrega tarea curso presencial):

1. Crear diagrama UML de clases.
2. Codificar cada una de las clases y testear salida.
3. Codificar main y testear cada una de los métodos de fabricación y retirada.

EXTRA (No necesario en la entrega de la tarea curso presencial):

4. Documentar generando javadoc.

APARTADO3.POO – HERENCIA.

APARTADO3.Ejercicio1

Crear una super clase llamada “Vehiculo” con los siguientes atributos y métodos:

- int speed;
- double regularPrice
- String color
- double getSalePrice();

Posteriormente, crea una subclase de la clase “Vehiculo” que se llama “Forgoneta”, esta clase tendrá los siguientes métodos y atributos propios:

- int weight
- double getSalePrice() // Si el peso > 2000 se le aplica un descuento del 10% sino 20%.

Nuevamente, crea una subclase de la clase “Vehiculo” nombrado “Ford”. Esta nueva clase tendrá los siguientes métodos y atributos propios:

- int year
- int manufacturerDiscount
- double getSalePrice() //Se utiliza el método de la superclase getSalePrice restándole el “manufacturerDiscount”.

Por último, crea una subclase llamada “Sedan” con los siguientes métodos y atributos propios:

- int longitud;
- double getSalePrice();//Si length>2 metros se aplica un descuento del 5%, sino 10%.

Cada una de las clases tendrán un constructor donde se le pasa todos los atributos.

Finalmente, crea una clase principal llamada “Concesionario” que contenga el método main.

1. Se creará una instancia de la clase “Sedan” e inicializa todos los atributos correspondientes (usando super para inicializar los atributos de la clase padre).
2. Crear dos instancias de la clase “Ford”
3. Crea una instancia de la clase “Coche”.
4. Muestra el precio “getSalePrice” para cada instancia.

APARTADO3.Ejercicio2

Nos piden hacer un programa que gestione empleados.

Los empleados se definen por tener: Nombre, Edad y Salario

También tendremos una constante llamada PLUS, que tendrá un valor de 300€

Luego tendremos dos tipos de empleados: repartidor y comercial.

1. El comercial, aparte de los atributos anteriores, tiene uno más llamado comisión (double).
2. El repartidor, aparte de los atributos de empleado, tiene otro llamado zona (String).

Crea sus constructores, getters and setters y toString (piensa como aprovechar la herencia).

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

Las clases tendrán un método llamado plus, que según en cada 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 aplicara el plus.
- En repartidor, si tiene menos de 25 y reparte en la "zona 3", este recibirá el plus.

En ambos casos en el método "plus" devolverá mediante un booleano si se le aplica o no el plus.

APARTADO3.Ejercicio2.2

En el ejercicio anterior vamos a probar el polimorfismo. Para ello crearemos un objeto llamado Pablo que empezará siendo comercial pero posteriormente cambia a repartido. ¿Cómo podemos implementar utilizando en ambos casos el mismo identificador?

Key: Utiliza el concepto de polimorfismo.

APARTADO3.Ejercicio3

Nos piden hacer que gestionemos una serie de productos. Para ello crearemos una clase padre llamada "Producto" que tiene los siguientes atributos: nombre y precio.

Luego 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 un método en todas las clases llamado "calcular()" que actualiza el atributo precio 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 al producto 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.

  Además, se indicará mostrando por pantalla que se ha actualiza el precio de un producto perecedero.

    - En NoPerecedero, El calculo que actualiza el atributo de precio hace lo mismo que en "Producto".

Además, se indicará mostrando por pantalla que se ha actualiza el precio de un producto no-perecedero.

Crea una clase main y crea un array/vector de 5 productos, posteriormente:

- Añadimos uno a uno 4 productos con diferentes días a caducar y 1 producto perecedero.
- Aplicamos para todos los productos registrados en el vector su método “calcular”
- Finalmente, mostramos su estado.

APARTADO3.Ejercicio4 (Herencia y matrices)

Nos piden hacer un almacén, vamos a usar programación orientado a objetos.

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

- ID, su identificador (cada uno tiene uno distinto): Numero entero entre 0 y 200.
- Cantidad de litros
- precio
- marca.

Además de los datos comunes para todas las bebidas:

- 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). Consideraremos 10 estanterías con 20 huecos.

Las operaciones del almacén son las siguientes:

- Calcular precio de todas las bebidas en el almacén: 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 en una estantería: agrega un producto en la primera posición libre de dicha estantería, si el identificador esta repetido en alguno de las bebidas, no se agregará esa bebida. O sino entra más productos en dicha estantería se devolverá un false. En caso de añadir la botella devolverá true.
- Eliminar un producto: dado un ID, eliminar el producto del almacén.
- Mostrar información: mostramos para cada bebida toda su información.

Puedes usar un main para probar las funcionalidades (añade productos, calcula precios, muestra información, etc)