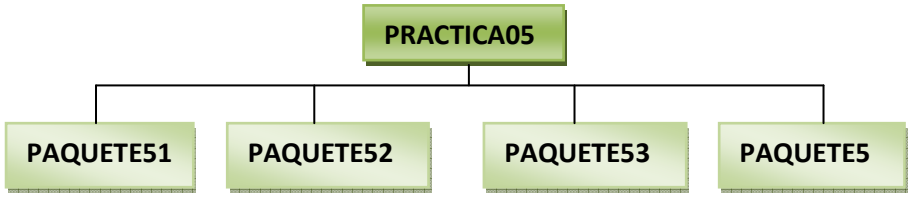
Práctica 5: Comienzo con clases, métodos, … (II)

Esta práctica la vamos a organizar en varios paquetes, para ello primero crearemos un proyecto llamado Practica05, y dentro de este proyecto 4 paquetes (Paquete51, ..,Paquete54), de forma que nos quedará algo como la siguiente esquema:



PAQUETE 5.1 (Clases, variables de instancia, métodos y constructores)

1. Crear la clase ***Semaforo***, cuyo estado se guarde en una propiedad llamado estadoSemaforo de tipo String que pueda tomar los valores "Verde", "Amarillo" y "Rojo", y que inicialmente tomará el valor "Rojo". Como métodos de acceso a la propiedad podríamos definir:
   - ✓ `void setColor(String Color)`. Cambia el color de la propiedad estadoSemaforo.
   - ✓ `String getColor()`. Devuelve el valor de la propiedad.

   Una vez creada la clase *Semaforo*, haremos otra clase llamada *PruebaSemaforo*, para probar el funcionamiento de la misma, esta nueva clase podría ser como la que se muestra a continuación:

```
class PruebaSemaforo
{
   public static void main(String[] args)
   {
          Semaforo miSemaforo = new Semaforo();
          Semaforo semaforoDeMiCalle = new Semaforo();
          Semaforo otroSemaforo = new Semaforo();
          miSemaforo.setColor("Rojo");
          otroSemaforo.setColor("Verde");
          System.out.println(otroSemaforo.getColor());
          System.out.println(semaforoDeMiCalle.getColor());

          if (miSemaforo.getColor().equals("Rojo"))
                System.out.println("No pasar");
   }
}
```

   Modificar la clase *Semaforo* y añadirle un constructor que inicialize el estado del semáforo al color que se especifique al crear un objeto de esa clase. Hacer otra clase de prueba, para comprobar el buen funcionamiento de la nueva clase semáforo.

2. Crear la clase ***Urna***, la cual contiene inicialmente un número determinado de bolas blancas y un número determinado de bolas negras, y presenta como interfaz los siguientes métodos:
   - ✓ **`Urna(int a, int b)`**. Constructor que inicializa las variables de instancia de la clase.
   - ✓ **`char sacaBola()`**. Devolverá el color de la bola sacada. Decrementa en 1 el nº de bolas de ese color.
   - ✓ **`void meteBola(char color).`** Incrementa en 1 el número de bolas del color dado.
   - ✓ **`logico quedanBolas()`**. Devuelve cierto si hay bolas en la urna.
   - ✓ **`logico quedaMasDeUnaBola()`**. Devuelve cierto si hay más de una bola en la urna.
   - ✓ **`int totalBolas()`**. Devuelve el número total de bolas (privado).
   - ✓ Métodos **`get`** y **`set.`**

- ✓ **`String toString().`** Este método devuelve una cadena con información sobre la urna, redefiniremos el método *toString* de la clase Object.

Utilizando un objeto de la clase anteriormente creada, realizar un programa para probarla que haga lo siguiente: mientras en la urna quede más de una bola, sacar dos bolas de la misma. Si ambas son del mismo color, introducir una bola negra en la urna. Si ambas son de distinto color, introducir una bola blanca en la urna. Extraer la última bola que queda y determinar su color.

3. Crear la clase ***Circulo*** con una variable miembro de tipo ***short*** que sea *radio*, y cuyo interfaz esté compuesto por:
   - ✓ Función constructora. Inicializa la variable miembro.
   - ✓ *`area()`*. Devuelve el área del círculo.
   - ✓ *`circunferencia()`*. Devuelve la circunferencia del círculo.
   - ✓ Métodos **`get, set`** y **`toString.`**
   - ✓ Dos métodos para poder comparar objetos de la clase Circulo, un método será de objeto y el otro de clase. Dos círculos serán iguales si coincide su radio. Estos métodos también serán una redefinición del método equals de la clase Object.

   Hacer otra clase en Java para comprobar el buen funcionamiento de la clase *Circulo*.

4. Crear la clase ***Rectangulo*** con dos variables miembro, *ancho* y *largo*, y cuyo interfaz esté compuesto por:
   - ✓ Función constructora. Inicializa las variables miembro, también llamadas de instancia.
   - ✓ *area()*. Devuelve el área del rectángulo.
   - ✓ *perimetro()*. Devuelve el perímetro del rectángulo.
   - ✓ Métodos **`get, set`** y **`toString.`**

   Hacer otra clase en Java para comprobar el buen funcionamiento de la clase *Rectángulo*.

5. Definir la clase ***Libro*** que guardará la siguiente información sobre los libros de una biblioteca:
   `autor: cadena`
   `titulo: cadena`
   `estado: lógico` (tomará el valor false si el libro está prestado y true si está en la biblioteca)

   Definir un interfaz a la clase libro con los siguientes métodos:
   - ✓ **`libro(cadena a, cadena t, logico e)`**. Constructor que inicializa los variables de instancia de un objeto de la clase libro.
   - ✓ Métodos **`get, set`** y **`toString.`**
   - ✓ Mëtodo **`equals`**, dos libros serán iguales si coincide su título y su autor.

   Una vez definido el interfaz, probarlo con el siguiente algoritmo:

```
class PruebaLibro
{
   public static void main(String[] args)
     {
      Libro b1=new Libro("Yerma","Federico Garcia Lorca",false);
      Libro b2=new Libro("Luces de Bohemia","Ram¢n del Valle Inclan",true);
      Libro b3=new Libro("Luces de Bohemia","Ram¢n del Valle Inclan",true);
      System.out.println(b1.toString());
      System.out.println(b2.toString());
```

```
        if (b1.getEstado()==false)
            System.out.println("El libro del objeto b1 esta prestado");
        else
            System.out.println("El libro del objeto b1 esta en la biblioteca");

        b1.setEstado(true);

        System.out.println(b1.toString());

        if (b1.equals(b2))
               System.out.println("Son iguales");
        else
               System.out.println("Son distintos");

        if (b2.equals(b3))
               System.out.println("Son iguales");
        else
               System.out.println("Son distintos");    }
}
```

6. Definir la clase **Coche** que guardará la siguiente información sobre los coches de una casa comercial:
   Modelo: cadena (privada)
   Precio sin iva: float (privada)
   Porcentaje de iva: byte (publica)

   Definir un interfaz a la clase coche con los siguientes métodos:

   - ✓ **`Coche(cadena m, float p)`**. Constructor de la clase.
   - ✓ **`void consulta()`**.Método que muestra el modelo de coche y su precio con iva.
   - ✓ Métodos **`get, set, equals`** y **`toString.`**
   - ✓ **`float precioReal()`**. Método que calcula el precio real del coche con iva incluido.

   Todos los métodos serán públicos, excepto el último (precioReal) que será un método privado. Una vez definido el interfaz, probarlo haciendo una clase de prueba.

   Modifica la clase PruebaCoche y llama al método precioReal para alguno de los objetos, ¿qué ocurre?. ¿Puedes cambiar el valor de las variables miembro *modelo* y *precio* desde la clase PruebaCoche? ¿Por qué puedes cambiar el valor de la variable *piva* desde PruebaCoche?

PAQUETE 5.2 (Clases, variables de instancia, métodos y constructores)

1. Crear la clase **DosContenedores** que mantiene información del contenido líquido de dos contenedores de tamaño dado en la inicialización. Teniendo en cuenta que ninguno de los contenedores, se puede llenar con más cantidad de líquido de la que realmente le cabe, dar métodos para:
   - ✓ *DosContenedores(int c1, int c2).* Crea dos contenedores del tamaño indicado y que inicialmente están vacíos.
   - ✓ *boolean llenarIzquierdo(int c).* Añade al contenedor izquierdo la cantidad indicada. Devuelve *true* si se ha podido añadir, y *false* en caso contrario.
   - ✓ *boolean llenarDerecho(int c).* Añade al contenedor derecho la cantidad indicada. Devuelve *true* si se ha podido añadir, y *false* en caso contrario.
   - ✓ *vaciarIzquierdo( ).* Deja vacío el contenedor izquierdo.
   - ✓ *vaciarDerecho( ).* Deja vacío el contenedor derecho.
   - ✓ *vaciarIzquierdosobreDerecho( ).* Vuelca el contenido del contenedor izquierdo sobre el contenedor derecho.

- ✓ *vaciarDerechoSobreIzquierdo( ).* Vuelca el contenido del contenedor derecho sobre el contenedor izquierdo.
- ✓ *estallenoIzquierdo( ).* Devuelve *true* si el contenedor izquierdo está lleno y *false* en caso contrario.
- ✓ *estallenoDerecho( ).* Devuelve *true* si el contenedor derecho está lleno y *false* en caso contrario.
- ✓ *estavacioIzquierdo( ).* Devuelve *true* si el contenedor izquierdo esta vacío y *false* en caso contrario.
- ✓ *estavacioDerecho( ).* Devuelve *true* si el contenedor derecho esta vacío y *false* en caso contrario.

A partir de la clase anterior diseña un programa que cree dos contenedores, el izquierdo con una capacidad de 5 litros y el derecho con una capacidad de 100 litros. Una vez creados se irán añadiendo litros hasta que alguno de los contenedores este lleno de la siguiente forma: al contenedor izquierdo se le irán añadiendo los litros de 1 en 1, mientras que para el derecho se seguirá la siguiente sucesión: 1, 2, 4, 8, 16, ... A continuación se volcará el contenido del contenedor izquierdo sobre el contenedor derecho, y finalmente se irá vaciando poco a poco el contenedor derecho volcando su contenido en el contenedor izquierdo y vaciando este último.

2. Crear la clase **CNP** que mantenga cuatro variables double ***x***, ***y***, ***z***, ***t*** con la siguiente interfaz:
   - ✓ `entrar(double)`. El valor de ***t*** se pierde, ***z*** pasa a ***t***, ***y*** pasa a ***z***, ***x*** pasa a ***y***, el valor del argumento pasa a ***x***.
   - ✓ `baja()`. ***y*** pasa a ***x***, ***z*** a ***y***, ***t*** a ***z***, ***t*** queda igual.
   - ✓ `xy()`. ***x*** cambia con ***y***.
   - ✓ `cambia()`. ***x*** cambia de signo.
   - ✓ `suma()`. ***y*** + ***x*** a ***x***, ***z*** a ***y***, ***t*** a ***z***, ***t*** queda igual.
   - ✓ `resta()`. ***y*** - ***x*** a ***x***, ***z*** a ***y***, ***t*** a ***z***, ***t*** queda igual.
   - ✓ `mult()`. ***y*** * ***x*** a ***x***, ***z*** a ***y***, ***t*** a ***z***, ***t*** queda igual.
   - ✓ `divi()`. ***y*** / ***x*** a ***x***, ***z*** a ***y***, ***t*** a ***z***, ***t*** queda igual.

   Utilizando la clase anteriormente creada hacer un programa que calcule la siguiente expresión:

   `(3+5*6)/(5-6*(3+1))`

3. a) Crear la clase **NodoLista** de forma que nos permita almacenar un dato de cualquier tipo y nos permita guardar una referencia a otro objeto de esta misma clase, con la siguiente interface:
      - ✓ `NodoLista(Object ob)`. Constructor de la clase NodoLista. Inicializa las variables de instancia.
      - ✓ `void ponSiguiente(NodoLista n)`. Enlaza el nodo n, al nodo del argumento implícito (*this).*
      - ✓ `Objetct elemento( )`. Devuelve el elemento almacenado en ese nodo.
      - ✓ `NodoLista siguiente( )`. *Devuelve el nodo siguiente.*

   b) Utilizando la clase NodoLista, anteriormente creada, construir la clase **Lista** con la siguiente interfaz:
      - ✓ `Lista()`. Constructor de la clase Lista. Crea una lista vacía.
      - ✓ `void ponPrimero(Object ob)`. Crea un nuevo nodo del tipo NodoLista, donde almacena *ob*, y lo coloca como primer nodo de nuestra lista.
      - ✓ `void ponUltimo(Objetct ob)`. Crea un nuevo nodo del tipo NodoLista, donde almacena *ob*, y lo coloca como último nodo de nuestra lista.
      - ✓ `boolean estaVacia()`. Devuelve verdadero si la lista está vacía y falso en caso contrario.

- ✓ `Objetct extraePrimero()`. Devuelve el primer elemento almacenado en la lista, eliminándolo de esta.
- ✓ `Object extraeUltimo()`. Devuelve el último elemento almacenado en la lista, eliminándolo de esta.
- ✓ `void visualizaElementos()`. Visualiza todos los elementos almacenados en la lista.

c) Realizar otra clase que pruebe el funcionamiento de las dos anteriores.

PAQUETE 5.3 (Variables de clase, Inicializadores, finalizadores, clases abstractas, herencia, interfaces)

1. Crear la clase **Cuenta** que mantiene la siguiente información por cada cuenta*: número de la cuenta, nombre del titular y saldo*. Además existen un *tipo de interés* que es común a todas las cuentas y que se mantiene siempre fijo en el 3% y un *acumulador del disponible* que hay en el banco en cada momento.

   El interfaz de esta clase debe proporcionar al menos:

   - ✓ *Inicializador estático:* Inicializa las variables de clase, si es que las hay.
   - ✓ *Constructor:* inicializa las variables miembro de objeto con los datos de creación de la cuenta.
   - ✓ *Ingreso:* Ingresa la cantidad indicada en la cuenta que se especifique.
   - ✓ *Reintegro:* Saca la cantidad indicada de la cuenta que se especifique.
   - ✓ Métodos **`get, set, equals`** y **`toString.`**
   - ✓ *Finalizador:* Mostrará un mensaje que diga "*Objeto destruido*".

   Hacer un algoritmo para probar el funcionamiento de la clase.

2. Construir las siguientes clases:

   - **Edificio:** almacena información sobre el número de habitaciones, el número de plantas y el área total de un determinado edificio.
   - **Casa:** contendrá la misma información que un edificio normal, además del número de dormitorios y el número de baños.
   - **Colegio:** contendrá la misma información que un edificio normal, además del número de clases y el número de oficinas.

   Las tres clases deben disponer, al menos, de un constructor, los métodos set y get y el método toString.

   A partir de las clases anteriores, crear un programa que cree un parque de bomberos que posee un área de 1000 metros repartidos en dos plantas y con un total de 10 habitaciones; una casa de dos plantas con una superficie total de 150 metros, repartidos en salón, cocina, 4 dormitorios, y 2 baños; y por último un colegio con un área de 2000 metros, 3 plantas, 25 habitaciones, 18 clases y 3 oficinas. Una vez creados los tres edificios, se deberá mostrar toda la información que se posea de cada uno de ellos.

3. A partir de la urna de ejercicios anteriores, desarrollar la clase **UrnaTrampa** con el siguiente comportamiento: cada vez que se saque una bola, hay una probabilidad del 0.2% de que una de las bolas que quedan dentro cambie de color (para ello ha de haber bolas de ambos colores en la urna). Probar el funcionamiento de la nueva clase.

4. Utilizando la clase Circulo creada anteriormente, crear la clase **Cilindro** con la siguiente interfaz:
   - ✓ Función constructora. Inicializa las variables radio y lado (generatriz).
   - ✓ `area()`. Devuelve el área del cilindro.
   - ✓ `volumen()`. Devuelve el volumen del cilindro.
   - ✓ Métodos **`get, set, equals`** y **`toString.`**

5. Utilizando la clase *Lista*, creada anteriormente, construir la clase **Pila** con la siguiente interface: *pop*, *push*, *vacia y cima.* Hacer un programa para comprobar el buen funcionamiento de dicha clase.

6. Utilizando la clase *Lista*, anteriormente creada, construir la clase **Cola** con la siguiente interface: *extraeDeCola*, *ponEnCola*, *vacia y frente.* Hacer un programa para comprobar el buen funcionamiento de dicha clase.

7. Vamos a crear una clase abstracta llamada **VehículoAbstracto** y dos clases derivadas: **Motocicleta** y **Camión**.

   La clase ***VehículoAbstracto*** contendrá las propiedades necesarias para albergar el color del vehículo (tipo *Color*), el número de ruedas que utiliza (byte), la cilindrada que tiene (short) y la potencia que ofrece (short). Estas propiedades serán *privadas* a la clase.

   Se definirán al menos los siguientes 6 constructores: 3 que sólo reciban como parámetro una de las 3 primeras propiedades, habrá uno por propiedad; otro al que le lleguen las propiedades color y número de ruedas, otro al que le lleguen las propiedades color, número de ruedas y cilindrada; y por último uno al que le lleguen las 4 propiedades.

   Y los métodos siguientes:

   - ✓ Métodos **`get, set`** y **`toString.`**
   - ✓ impuesto: método abstracto que será definido en cada subclase derivada, devolverá un valor *float*.

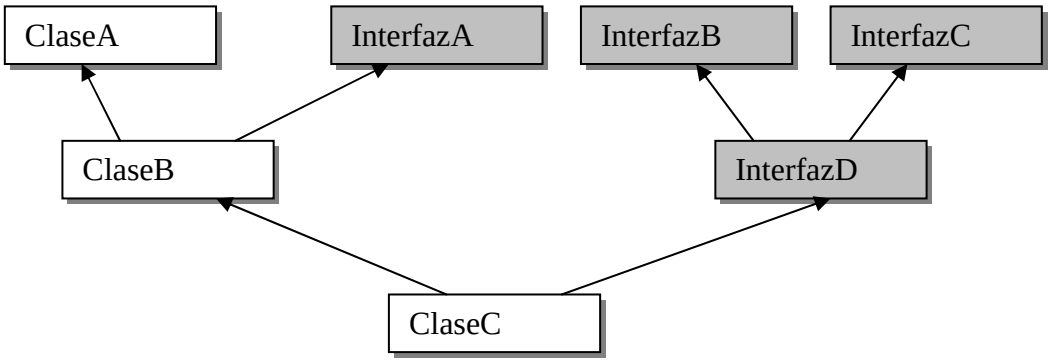
   La clase derivada ***Camión*** establecerá una nueva propiedad que indique el número de ejes que posee el camión (byte), junto a los siguientes constructores: uno al que le llegue sólo el número de ruedas; otro al que le llegue el color y número de ruedas; otro al que le llegue el color, número de ruedas y cilindrada; otro al que le llegue color, número de ruedas, cilindrada y potencia; y otro al que lleguen las cinco propiedades de un camión. Además ofrecerá los siguientes métodos:

   - ✓ Métodos **`get, set`** y **`toString.`**
   - ✓ impuesto: suponemos que el impuesto que pagan los camiones se calcula realizando el sumatorio de los términos: cilindrada/30, potencia*20, número de ruedas*20 y número de ejes*50.

   La clase derivada ***Motocicleta*** contendrá una nueva propiedad que albergue el número de ocupantes permitido que puede transportar cada motocicleta (byte), además sabemos que sólo tiene 2 ruedas. Tendrá los siguientes constructores: uno al que no le llegue ningún parámetro, y que llama al constructor de la clase VehiculoAbstracto indicando el número de ruedas; otro al que le llegue sólo el número de plazas; otro al que le llegue sólo el color; otro al que le llegue el color y la cilindrada; y por ultimo uno al que le llegue el color, la cilindrada y la potencia. Además ofrecerá los siguientes métodos:

   - ✓ Métodos **`get, set`** y **`toString.`**
   - ✓ impuesto: suponemos que el impuesto que pagan las motocicletas se calcula aplicando la fórmula: cilindrada/30 + potencia*30.

8. Definir un conjunto de clases e interfaces que se relacionen como se muestra a continuación, con los miembros que consideres oportunos.



PAQUETE 5.4

1. Construir una clase ***Estadistica*** que mantenga información de:
   - ✓ $\sum n_i$
   - ✓ $\sum n_i x_i$
   - ✓ $\sum n_i x_i^2$

   El interfaz que debe ofrecer es:

   - `agrega(v)`: agrega un elemento *v* a la estadística.
   - `agrega(v,n)`: agrega *n* veces un elemento *v* a la estadística.
   - `media( )`: devuelve la media aritmética de los valores.
   - `desviacionTipica( )`: devuelve la desviación típica.
   - `inicializa( )`: inicializa todas las variables.

   Realizar también una clase ***MainEstadistica*** para probar el funcionamiento de la clase Estadística.

2. Se pretende generar una lista enlazada de números primos desde el 2 hasta un número dado como tope.

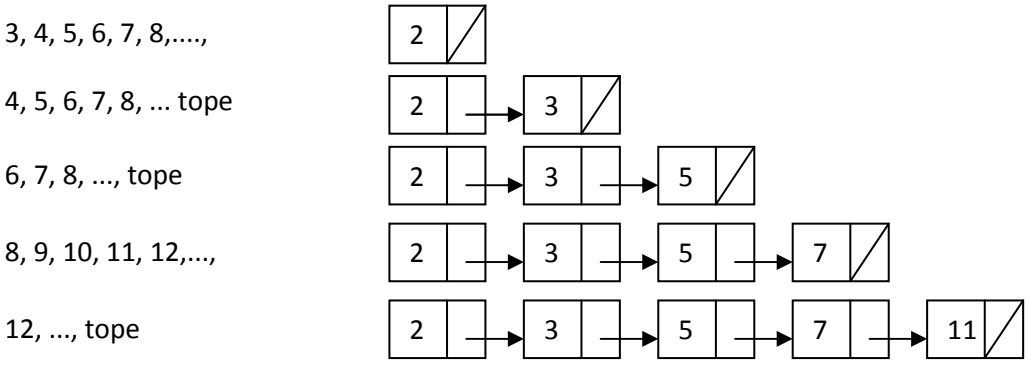
   Para ello:

   - Construimos inicialmente un nodo que contiene al número 2 y una referencia (inicialmente nula) a otro posible nodo. Posteriormente, partiendo del nodo que contiene el número 2, iteramos desde el 3 hasta el número dado como tope realizando lo siguiente:

     *Por cada número de la iteración, llamémosle **x**, por ejemplo, habrá que recorrer la lista que se irá generando de números primos, si encontramos algún nodo cuyo número sea múltiplo de **x**, sabremos que no es primo.*

     *En otro caso si se recorre la lista de nodos y ninguno de ellos es múltiplo de **x**, habrá que añadir **x** al final de la lista de nodos con números primos.*

   - Gráficamente se irá produciendo algo similar a lo siguiente:



- Se pide:

  Realizar la clase ***Primo*** que tenga como variables de instancia las dos necesarias de cada nodo (objeto), y al menos los siguientes métodos:

  - `Primo(n)`: Constructor, crea un nodo (objeto) con un número primo *n*.
  - `AddPrimo(n)`: Añade un nuevo nodo al final de la lista con el número primo *n*.
  - `numPrimo()`. Devuelve el número de un nodo de la clase Primo.
  - `sigPrimo()`. Devuelve una referencia al nodo siguiente al actual.

  Realizar la clase ***MainPrimo*** que permita generar una lista de valores primos hasta un tope dado, como se ha explicado al inicio del ejercicio. Este programa deberá después recorrer la lista enlazada de primos y mostrarlos por pantalla.

3. Crear la clase **Urna**, la cual contiene inicialmente un número indeterminado de bolas blancas y de bolas negras y presenta como interfaz los siguientes métodos: *sacaBola( ), meteBola(color), quedanBolas( ), quedaMasDeUnaBola( ), totalBolas( ).*

   Utilizando la clase anteriormente creada, crear la clase **UrnaDeTresColores** (negras, blancas y rojas). Con dicha clase realizar un programa que haga lo siguiente: mientras en la urna quede más de una bola, sacar dos bolas de la misma. Si ambas son blancas, introducir una bola roja en la urna. Si ambas son iguales, menos blancas, introducir una bola del mismo color en la urna. Y si son distintas se mete una bola blanca. Extraer la última bola que queda y determinar su color.