POO en Java

Ejercicio 1

Implementa la clase `Caballo` vista en un ejercicio anterior. Pruébala creando instancias y aplicándole algunos métodos.

Ejercicio 2

Crea la clase `Vehiculo`, así como las clases `Bicicleta` y `Coche` como subclases de la primera. Para la clase `Vehiculo`, crea los atributos de clase `vehiculosCreados`y `kilometrosTotales`, así como el atributo de instancia `kilometrosRecorridos`. Crea también algún método específico para cada una de las subclases. Prueba las clases creadas mediante un programa con un menú como el que se muestra a continuación:

```
VEHÍCULOS
=========
1. Anda con la bicicleta
2. Haz el caballito con la bicicleta
3. Anda con el coche
4. Quema rueda con el coche
5. Ver kilometraje de la bicicleta
6. Ver kilometraje del coche
7. Ver kilometraje total
8. Salir
Elige una opción (1-8):
```

Ejercicio 3

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

Ejercicio 4

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

Ejercicio 5

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
```

Ejercicio 6

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.

Ejercicio 7

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.

La clase Zona con sus atributos y métodos se muestra a continuación:

```
public class Zona {

  private int entradasPorVender;

  public Zona(int n){
    entradasPorVender = n;
  }

  public int getEntradasPorVender() {
    return entradasPorVender;
  }

  /**
   * Vende un número de entradas.
   * Comprueba si quedan entradas libres antes de realizar la venta.
   *
   * @param n número de entradas a vender
   */
  public void vender(int n) {

    if (this.entradasPorVender == 0) {
      System.out.println("Lo siento, las entradas para esa zona están agotadas.");
    } else if (this.entradasPorVender < n) {
      System.out.println("Sólo me quedan " + this.entradasPorVender
                         + " entradas para esa zona.");
    }

    if (this.entradasPorVender >= n) {
      entradasPorVender -= n;
      System.out.println("Aquí tiene sus " + n + " entradas, gracias.");
    }
  }
}
```

El menú del programa debe ser el que se muestra a continuación. Cuando elegimos la opción 2, se nos 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. Mostrar número de entradas libres
2. Vender entradas
3. Salir
```

Ejercicio 8

Implementa la clase `Terminal`. Un terminal tiene asociado un número. Los terminales se pueden llamar unos a otros y el tiempo de conversación corre para ambos. A continuación se proporciona el contenido del `main` y el resultado que debe aparecer por pantalla.

Programa principal:

```
Terminal t1 = new Terminal("678 11 22 33");
Terminal t2 = new Terminal("644 74 44 69");
Terminal t3 = new Terminal("622 32 89 09");
Terminal t4 = new Terminal("664 73 98 18");
System.out.println(t1);
System.out.println(t2);
t1.llama(t2, 320);
t1.llama(t3, 200);
System.out.println(t1);
System.out.println(t2);
System.out.println(t3);
System.out.println(t4);
```

Salida:

```
Nº 678 11 22 33 - 0s de conversación
Nº 644 74 44 69 - 0s de conversación
Nº 678 11 22 33 - 520s de conversación
Nº 644 74 44 69 - 320s de conversación
Nº 622 32 89 09 - 200s de conversación
Nº 664 73 98 18 - 0s de conversación
```

Ejercicio 9

Implementa la clase `Movil` como subclase de `Terminal` (la clase del ejercicio anterior que ya no hace falta modificar). 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. A continuación se proporciona el contenido del main y el resultado que debe aparecer por pantalla. Para que el total tarificado aparezca con dos decimales, puedes utilizar `DecimalFormat`.

Programa principal:

```
Movil m1 = new Movil("678 11 22 33", "rata");
Movil m2 = new Movil("644 74 44 69", "mono");
Movil m3 = new Movil("622 32 89 09", "bisonte");
System.out.println(m1);
System.out.println(m2);
m1.llama(m2, 320);
m1.llama(m3, 200);
m2.llama(m3, 550);
System.out.println(m1);
System.out.println(m2);
System.out.println(m3);
```

Salida:

```
Nº 678 11 22 33 - 0s de conversación - tarificados 0,00 euros
Nº 644 74 44 69 - 0s de conversación - tarificados 0,00 euros
Nº 678 11 22 33 - 520s de conversación - tarificados 0,52 euros
Nº 644 74 44 69 - 870s de conversación - tarificados 1,10 euros
Nº 622 32 89 09 - 750s de conversación - tarificados 0,00 euros
```

Ejercicio 10

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`, 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. Nótese que el método `come` está sobrecargado.

Programa principal:

```
Ameba a1 = new Ameba();
a1.come(2);
System.out.println(a1);
Ameba a2 = new Ameba();
a2.come(4);
System.out.println(a2);
a1.come(a2);
System.out.println(a1);
System.out.println(a2);
a2.come(3);
System.out.println(a2);
```

Salida:

```
Soy una ameba y peso 4 microgramos.
Soy una ameba y peso 6 microgramos.
Soy una ameba y peso 9 microgramos.
Soy una ameba y peso 0 microgramos.
Soy una ameba y peso 2 microgramos.
```

Ejercicio 11

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. 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.

Programa principal:

```
TarjetaRegalo t1 = new TarjetaRegalo(100);
TarjetaRegalo t2 = new TarjetaRegalo(120);
System.out.println(t1);
System.out.println(t2);
t1.gasta(45.90);
t2.gasta(5);
t2.gasta(200);
t1.gasta(3.55);
System.out.println(t1);
System.out.println(t2);
TarjetaRegalo t3 = t1.fusionaCon(t2);
System.out.println(t1);
System.out.println(t2);
System.out.println(t3);
```

Salida:

```
Tarjeta nº 67324 - Saldo 100.00€
Tarjeta nº 02788 - Saldo 120.00€
No tiene suficiente saldo para gastar 200.00€
Tarjeta nº 67324 - Saldo 50.55€
Tarjeta nº 02788 - Saldo 115.00€
Tarjeta nº 67324 - Saldo 0.00€
Tarjeta nº 02788 - Saldo 0.00€
Tarjeta nº 59032 - Saldo 165.55€
```

Ejercicio 12

Se quiere informatizar una biblioteca. Crea las clases **Publicacion**, **Libro** y **Revista**. Las clases deben estar implementadas con la jerarquía correcta. Las características comunes de las revistas y de los libros son el código ISBN, el título, y el año de publicación. Los libros tienen además un atributo prestado. Cuando se crean los libros, no están prestados. Las revistas tienen un número. La clase **Libro** debe implementar la interfaz **Prestable** que tiene los métodos **presta**, **devuelve** y **estaPrestado**.

Programa principal:

```
Libro libro1 = new Libro("123456", "La Ruta Prohibida", 2007);
Libro libro2 = new Libro("112233", "Los Otros", 2016);
Libro libro3 = new Libro("456789", "La rosa del mundo", 1995);
Revista revista1 = new Revista("444555", "Año Cero", 2019, 344);
Revista revista2 = new Revista("002244", "National Geographic", 2003, 255);
System.out.println(libro1);
System.out.println(libro2);
System.out.println(libro3);
System.out.println(revista1);
System.out.println(revista2);
libro2.presta();
if (libro2.estaPrestado()) {
  System.out.println("El libro está prestado");
}
libro2.presta();
libro2.devuelve();
if (libro2.estaPrestado()) {
  System.out.println("El libro está prestado");
}
libro3.presta();
System.out.println(libro2);
System.out.println(libro3);
```

Salida:

```
ISBN: 123456, título: La Ruta Prohibida, año de publicación: 2007 (no prestado)
ISBN: 112233, título: Los Otros, año de publicación: 2016 (no prestado)
ISBN: 456789, título: La rosa del mundo, año de publicación: 1995 (no prestado)
ISBN: 444555, título: Año Cero, año de publicación: 2019
ISBN: 002244, título: National Geographic, año de publicación: 2003
El libro está prestado
Lo siento, ese libro ya está prestado.
ISBN: 112233, título: Los Otros, año de publicación: 2016 (no prestado)
ISBN: 456789, título: La rosa del mundo, año de publicación: 1995 (prestado)
```

Ejercicio 13

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. En el siguiente capítulo se propone un ejercicio como mejora de éste, en el que se pide llevar un registro de los movimientos realizados.

Programa principal:

```
CuentaCorriente cuenta1 = new CuentaCorriente();
CuentaCorriente cuenta2 = new CuentaCorriente(1500);
CuentaCorriente cuenta3 = new CuentaCorriente(6000);
System.out.println(cuenta1);
System.out.println(cuenta2);
System.out.println(cuenta3);
cuenta1.ingreso(2000);
cuenta2.cargo(600);
cuenta3.ingreso(75);
cuenta1.cargo(55);
cuenta2.transferencia(cuenta3, 100);
System.out.println(cuenta1);
System.out.println(cuenta2);
System.out.println(cuenta3);
```

Salida:

```
Número de cta: 6942541557 Saldo: 0,00 €
Número de cta: 9319536518 Saldo: 1500,00 €
Número de cta: 7396941518 Saldo: 6000,00 €
Número de cta: 6942541557 Saldo: 1945,00 €
Número de cta: 9319536518 Saldo: 800,00 €
Número de cta: 7396941518 Saldo: 6175,00 €
```

Ejercicio 14

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 proporcionan ambos valores. 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]. A continuación se proporciona el contenido del main y el resultado que debe aparecer por pantalla.

Programa principal:

```
FichaDomino f1 = new FichaDomino(6, 1);
FichaDomino f2 = new FichaDomino(0, 4);
FichaDomino f3 = new FichaDomino(3, 3);
FichaDomino f4 = new FichaDomino(0, 1);
System.out.println(f1);
System.out.println(f2);
System.out.println(f3);
System.out.println(f4);
```

```
System.out.println(f2.voltea());
System.out.println(f2.encaja(f4));
System.out.println(f1.encaja(f4));
System.out.println(f1.encaja(f3));
System.out.println(f1.encaja(f2));
```

Salida:

```
[6|1]
[ |4]
[3|3]
[ |1]
[4| ]
true
true
false
false
```

Ejercicio 15

Utiliza la clase anterior para generar una secuencia de 8 fichas creadas de forma aleatoria, que encajen bien y que estén bien colocadas según el juego del dominó. No hay que controlar si se repiten o no las fichas.

Ejemplo:

```
[6|1][1|4][4|4][4| ][ |3][3|2][2|6][6|5]
```

Ejercicio 16

Crea las clases **Punto** y **Linea**. De un punto se tienen que saber sus coordenadas x e y, mientras que una línea está definida por dos puntos. Define las clases y los métodos necesarios para que el siguiente código muestre la salida que se indica.

Programa principal:

```
Punto p1 = new Punto(4.21, 7.3);
Punto p2 = new Punto(-2, 1.66);
Linea l = new Linea(p1, p2);
System.out.println(l);
```

Salida:

```
Línea formada por los puntos (4.21, 7.3) y (-2.0, 1.66)
```

Ejercicio 17

Implementa las clases **Piramide** y **Rectangulo**. Sobre una pirámide se debe saber su **altura** y sobre un rectángulo se debe saber tanto la **base** como la **altura**. Cada una de las clases debe tener un atributo de clase (static) que lleve la cuenta de las pirámides y de los rectángulos creados respectivamente. El siguiente código que va dentro del `main` genera la salida que se indica.

Programa principal:

```
Piramide p = new Piramide(4);
Rectangulo r1 = new Rectangulo(4, 3);
Rectangulo r2 = new Rectangulo(6, 2);
System.out.println(p);
System.out.println(r1);
System.out.println(r2);
System.out.println("Pirámides creadas: " + Piramide.getPiramidesCreadas());
System.out.println("Rectángulos creados: " + Rectangulo.getRectangulosCreados());
```

Salida:

```
   *
  ***
 *****
*******

****
****
****

******
******

Pirámides creadas: 1
Rectángulos creados: 2
```

Ejercicio 18

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). 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”**. El siguiente trozo de código que va dentro del `main` genera la salida que se muestra a continuación.

Programa principal:

```
Incidencia inc1 = new Incidencia(105, "No tiene acceso a internet");
Incidencia inc2 = new Incidencia(14, "No arranca");
Incidencia inc3 = new Incidencia(5, "La pantalla se ve rosa");
Incidencia inc4 = new Incidencia(237, "Hace un ruido extraño");
Incidencia inc5 = new Incidencia(111, "Se cuelga al abrir 3 ventanas");
inc2.resuelve("El equipo no estaba enchufado");
inc3.resuelve("Cambio del cable VGA");
System.out.println(inc1);
System.out.println(inc2);
System.out.println(inc3);
System.out.println(inc4);
System.out.println(inc5);
System.out.println("Incidencias pendientes: " + Incidencia.getPendientes());
```

Salida:

```
Incidencia 1 - Puesto: 105 - No tiene acceso a internet - Pendiente
Incidencia 2 - Puesto: 14 - No arranca - Resuelta - El equipo no estaba enchufado
Incidencia 3 - Puesto: 5 - La pantalla se ve rosa - Resuelta - Cambio del cable VGA
Incidencia 4 - Puesto: 237 - Hace un ruido extraño - Pendiente
Incidencia 5 - Puesto: 111 - Se queda colgado al abrir 3 ventanas - Pendiente
Incidencias pendientes: 3
```