Ejercicios herencia.

1. Aulas

Diseñar una jerarquía de clases para representar las **aulas** de un centro de estudios.

De un aula se conoce el código (numérico), la longitud y la anchura. Se desea un método que devuelva la superficie del aula y otro que devuelva su capacidad. La capacidad se calcula a partir de la superficie a razón de 1 alumnos por cada 1.4 metros cuadrados de superficie.

Además de las aulas, digamos *normales*, existen aulas de **informática** y aulas de **música:**

- Un dato de las aulas de música es si tienen o no piano. A la hora de calcular su capacidad, hay que tener en cuenta que el piano (si está presente) hace que quepan dos personas menos.
- De las aulas de informática se conoce el número de ordenadores y su capacidad no se calcula en función de la superficie, sino a razón de dos alumnos por ordenador.

Implementar también el método toString de cada una de las clases diseñadas para que

devuelva:

- En las aulas normales, el código y la superficie y la capacidad.
- En las aulas de música, se añadirá la información de si hay o no piano.
- En las aulas de informática, se añadirá la información del número de ordenadores

2. Juegos

Un salón de videojuegos dispone de ordenadores en las que los clientes pueden jugar. Además de jugar en el establecimiento, la empresa alquila y vende juegos.

a) Diseñar la clase Juego siguiendo las siguientes especificaciones:

- Atributos protected: título (String), fabricante (String), año (int).
- Constructor public Juego(String t, String f, int a)
- Consultores y modificadores de todos los atributos
- public String toString(), que devuelve un String con los datos del Juego
- public boolean equals (Object o): Dos juegos son iguales si tienen el mismo título, fabricante y año.
- public int compareTo (Juego o): Un juego es menor que otro si su título es menor. A igual título, si su fabricante es menor. A igual título y fabricante, si su año es menor.

b) Diseñar las clases JuegoEnAlquiler y JuegoEnVenta sabiendo que, además de los atributos descritos anteriormente, tienen.

- precio
- nº de copias disponibles

· JuegoEnAlquiler

- tiene un atributo que indica el número de días que se alquila
- Constructor que recibe todos sus datos
- Método alquilar que decrementa el número de copias disponibles, siempre y cuando sea mayor que 0. Devolverá un boolean indicando si el alquiler ha sido posible o no (número de copias disponibles mayor o no que cero)
- Método devolver que incrementa el número de copias disponibles.
- toString(): String con todos los datos del JuegoEnAlquiler

· JuegoEnVenta

- Constructor que recibe todos sus datos

- tiene un método vender, que decrementa el número de copias disponibles , siempre y cuando sea mayor que 0. Devolverá un boolean indicando si la venta ha sido posible o no (número de copias disponibles mayor o no que cero)
- toString(): String con todos los datos del JuegoEnVenta

c) Las clases JuegoEnAlquiler y JuegoEnVenta tienen atributos comunes y métodos con comportamiento similar. Diseña la clase JuegoAlquilerVenta que herede de Juego y haz que JuegoEnAlquier y JuegoEnVenta hereden de ella. JuegoAlquilerVenta debe representar todo lo común a estas dos clases.

3. Figuras

Diseñar la clase Figura y sus subclases Circulo y Rectangulo, teniendo en cuenta las siguientes especificaciones:

Toda figura tiene:

- Una posición en el eje X y una posición en el eje Y (int).
- Un color (String)

Los círculos, además

- Un radio (int)

Los rectángulos

- Una altura y una anchura (int)

Implementar las clases con sus respectivos constructores y los siguientes métodos:

- area, que devuelve la superficie de la figura.
- toString: String con el color, posición y área de la figura. (Implementarlo solo en la clase Figura)
- equals: Dos figuras son iguales si son del mismo tipo y tienen mismo color, posición y tamaño (mismo radio en los círculos, misma altura y anchura en los rectángulos).

4.- Monedas y billetes

La Fabrica Nacional de Moneda y Timbre quiere almacenar cierta información técnica del dinero (billetes y monedas) que emite. En concreto, le interesa:

- Valor: Valor de la moneda o billete ,en euros.(double)
- Año de emisión: Año en que fué emitida la moneda o billete. (int)
- De las monedas,
    - Diámetro: Diámetro de la moneda, en milímetros. (double)
    - Peso: Peso de la moneda, en gramos (double)
- De los billetes.
    - Altura del billete, en mm (double)
    - Anchura del billete, en mm (double).

a) Diseñar la clase abstracta Dinero y sus subclases Moneda y Billete, desarrollando:

- Constructores que reciban los datos de cada clase
- equals: Dos monedas o billetes son iguales si tienen el mismo año de emisión y valor.
- compareTo: Es menor el de menor año, a igual año es menor el de menor valor.
- toString: Que muestre todos los datos del billete o moneda. Los billetes irán precedidos por el texto “BILLETE” y las monedas por el texto “MONEDA”

b) Diseñar la clase TestDinero para probar las clases desarrolladas: Crear objetos de las clases Moneda y Billete y mostrarlos por pantalla.

5.- Televisores

Un centro comercial quiere mostrar cierta información sobre los televisores que vende. Los televisores pueden ser de dos tipos: de tubo o LCD. En concreto, de cada televisor le interesa mostrar

- Marca (String)
- Modelo (String)
- Precio en euros
- Pulgadas de la pantalla (double).
- Resolución: La resolución se mide de forma distinta en los televisores de tubo que en los televisores LCD. En los TV de tubo se mide en **lineas**. En los TV LCD se mide **pixels horizontales** x **pixels verticales**.

a) Diseñar la clase Televisor con los atributos y métodos comunes a los dos tipos de televisores y sus subclases TVTubo y TVLCD con los atributos y métodos que sea necesario:

- Constructor de cada clase que permita inicializar todos los datos de la clase.
- equals: Dos televisiones son iguales si son de la misma marca y modelo.
- compareTo: Se considera menor (mayor) la de menor (mayor) marca. A igual marca, menor (mayor) la de menor (mayor) modelo.
- public String resulucion(): Devuelve un texto con la resolución del televisor, como por ejemplo “420 lineas” o “800 x 600 pixels” dependiendo del tipo de televisor.
- public String toString(): Devuelve un texto con la marca, modelo, precio, pulgadas y resolución.

b) Diseñar la clase TestTV para probar las clases diseñadas. Crear algunos objetos de las clases TVTubo y TVLCD y mostrarlos por pantalla.

5.- Test

De cada pareja de afirmaciones indica cual es la verdaderas:

a) Se dice que instanciamos una clase cuando creamos objetos de dicha clase.
b) Se dice que instanciamos una clase cuando creamos una subclase de dicha clase.

c) Si una clase es abstracta no se puede instanciar.
d) Si una clase es abstracta no se puede heredar de ella.

e) Una clase abstracta tiene que tener métodos abstractos.
f) Una clase puede ser abstracta y no tener métodos abstractos.

g) Si una clase tiene métodos abstractos tiene que ser abstracta.
h) Una clase puede tener métodos abstractos y no ser abstracta.

i) Si una clase es abstracta sus subclases no pueden ser abstractas.
j) Una clase abstracta puede tener subclases que también sean abstractas.

k) Si un método es abstracto en una clase, tiene que ser no abstracto en la subclase, o bien, la subclase tiene que ser también abstracta.
l) Si un método es abstracto en una clase, no puede ser abstracto en las subclases.

m) Si un método se define *final* se tiene que reescribir en las subclases.
n) Si un método se define *final* no se puede reescribir en las subclases

o) Una clase puede tener un método *final* y no ser una clase *final*
p) Si una clase tiene un método *final* tiene que ser una clase *final*

q) Si una clase se define *final* no se pueden definir subclases de ella.
r) Si una clase se define *final* no se puede instanciar.

s) Un método definido *final* y *abstract* resultaría inútil, puesto que nunca se podría implementar en las subclases.
t) Un método definido *final* y *abstract* podría resultar útil.

6. Dada las siguientes definiciónes de clases:

```
public class Persona {
	private String nombre;
	private int edad;
	public Persona (){
		this.nombre = "";
		this.edad = 0;
	}
	public Persona(String n, int e){
		this.nombre = n;
		this.edad = e;
	}
	public String toString(){
		return "Nombre: " + nombre + "Edad " + edad;
	}
	public final String getNombre (){
		return nombre;
	}
	public final int getEdad(){
		return edad;
	}
}
//-----------------------------------------------------------------
class Estudiante extends Persona {
	private double creditos;
	public Estudiante(String n, int e, double c){
		super(n,e);
		this.creditos = c;
	}
	public String toString(){
		return super.toString() +"\nCreditos: "+ creditos;
	}
}
//-----------------------------------------------------------------
class Empleado extends Persona {
	private double salario;
	public Empleado(String n, int e, double s){
		super(n,e);
		this.salario = s;
	}
	public String toString(){
		return "Nombre: "+ nombre + "\nSalario: "+ salario;
	}
}
//-----------------------------------------------------------------
class Test{
	public static void main(String[] args) {
		Estudiante e = new Estudiante("pepe",18,100);
		System.out.println(e.toString());
	}
}
```

Responde a las siguientes cuestiones justificando las respuestas.

- ¿Es necesario el uso de this en el contructor de la clase Estudiante?
- ¿Es necesario el uso de super en el método toString de la clase Estudiante?
- Si quitásemos el constructor de la clase Estudiante ¿daría un error de compilación?
- En el método toString de la clase Empleado ¿por qué es incorrecto el acceso que se hace al atributo nombre? ¿Cómo se tendría que definir nombre en la clase Persona para evitar el error?
- ¿Qué consecuencia tiene que algunos métodos de la clase Persona se hayan definido final?
- Si el método toString no se hubiera definido en ninguna de las tres clases ¿daría error el System.out.println del método main?

7.- Amarres

En un puerto se alquilan amarres para barcos de distinto tipo. De cada amarre se guarda la posición del amarre, el cliente que lo tiene alquilado, el número de días que el amarre estará alquilado y el barco que lo ocupará.

De los clientes se guarda el nombre y el dni.

Un barco se caracteriza por su matrícula, su eslora en metros y año de fabricación.

El importe del alquiler se calcula multiplicando el número de días de ocupación por un factor, que depende de cada barco.

Hay tres subtipo barcos: Veleros, Deportivos y Yates.

- De los veleros se conoce el número de mástiles.
- De los barcos deportivos, la potencia.
- De los yates, la potencia y el número de camarotes.

El factor de cada barco se calcula de la siguiente manera:

- En general, para todos los barcos, será 2 multiplicado por los metros de eslora.
- A los veleros se sumará a este factor el número de mástiles.
- A los barcos deportivos se sumará a este factor la potencia.
- Para los yates el factor no depende de los metros de eslora y se calcula multiplicando la potencia por el número de camarotes.

Diseñar las clases Cliente, Amarre, Barco, Deportivo, Yate y Velero con los atributos y métodos necesarios (constructores, getter, setter, toString, equals …). La clase Amarre tendrá un método calcularPrecioAlquiler. La clase Barco y sus subclases tendrán un método getFactorAlquiler.