FAMILIA PROFESIONAL:	**Informática y Comunicaciones**
CICLOS FORMATIVOS:	**Desarrollo de Aplicaciones Multiplataforma, Desarrollo de Aplicaciones Web**
MÓDULO:	**Programación**

UNIDAD 6: RELACIONES ENTRE CLASES Y POLIMORFISMO

ACTIVIDADES

AUTORES: **Fernando Rodríguez Alonso**
Sonia Pasamar Franco

ASOCIACIONES ENTRE CLASES

ACTIVIDAD 1x01

Codifica una clase **`Videojuego`** para tratar la información de los diferentes videojuegos de una tienda. De cada uno de ellos se desea guardar: un código autogenerado, el título, el precio en euros y la cantidad que se tiene de ese videojuego en *stock*.

El código de cada videojuego será un número entero correlativo. Cuando la tienda saca un nuevo videojuego a la venta, se le asignará el código siguiente al código del último videojuego que se creó. Si la tienda deja de vender un videojuego, el código de ese videojuego no se reutilizará para otro videojuego.

Para esta clase **`Videojuego`**:

- Codifica un constructor que reciba como parámetros el título y el precio. Al instanciar un objeto de esta clase, la cantidad de unidades del videojuego será 10.
- Codifica un método de objeto que devuelva una cadena de texto con el resumen de los atributos del objeto. Para ello, sobrescribe el método **`toString`** que procede de la clase `Object`. El precio se deberá indicar con 2 dígitos decimales.

Codifica una clase **`Almacen`** que permita gestionar un contenedor de videojuegos. Además de un vector de videojuegos, esta clase contendrá un índice que tendrá una doble función: marcará la tanto la primera posición libre del vector de videojuegos, como el número de elementos que éste contiene instanciados.

Para esta clase **`Almacen`**:

- Codifica un constructor que reciba como parámetro el número de videojuegos que puede contener (es decir, la capacidad del vector). Al instanciar un objeto de esta clase, el índice del almacén será 0.
- Codifica un método de objeto que devuelva una cadena de texto con el resumen de cada uno de los videojuegos, precedido por su posición dentro del vector, que será el valor que el usuario indique cuando quiera acceder a un videojuego. Para ello, sobrescribe el método **`toString`** que procede de la clase `Object`.
  Si no hay videojuegos en el vector, este método devolverá el mensaje:
  ```
  El almacén está vacío.
  ```
  Si el vector contiene videojuegos, este método devolverá un mensaje como el siguiente:
  ```
  (0) Videojuego[Código=1, Título=BofW, Precio=49.95, Cantidad=10]
  (1) Videojuego[Código=2, Título=GTA5, Precio=19.95, Cantidad=10]
  (2) Videojuego[Código=3, Título=FH4, Precio=29.95, Cantidad=10]
  ```
- Codifica un método de objeto **`consultar`** que tenga como parámetro la posición del videojuego a consultar y que devuelva el videojuego que se encuentra en dicha posición en el vector o `null` si no hay ningún videojuego en esa posición.
- Codifica un método de objeto **`insertar`** que tenga como parámetro un videojuego a insertar, que realice la operación cuando el vector no esté lleno, y que devuelva un booleano indicando si la inserción se ha realizado con éxito o no.
- Codifica un método de objeto **`eliminar`** que tenga como parámetro la posición del videojuego a eliminar, que realice la operación cuando la posición referencie a un videojuego en el vector, y que devuelva un booleano indicando si la eliminación se ha realizado con éxito o no.

Codifica una clase **`Actividad_1x01`** que incluya un programa principal **`main`** que gestione un almacén de videojuegos (con capacidad para 20 videojuegos) con el menú de opciones siguiente:

0) Salir del programa.

1) Insertar un videojuego en el almacén.

Leerá por teclado el título y el precio del videojuego a insertar.
Intentará realizar la inserción del videojuego, creado a partir de los datos leídos desde teclado, en el almacén.
Si el almacén está lleno (con 20 videojuegos), visualizará en consola el mensaje:

```
Error al insertar: almacén lleno.
```

Si la inserción se ha realizado con éxito, visualizará en consola el mensaje:

```
Se ha insertado el videojuego en el almacén con éxito.
```

2) Eliminar un videojuego, por posición, del almacén.

Leerá por teclado la posición del videojuego a eliminar.
Intentará realizar la eliminación del videojuego, indicado mediante la posición leída desde teclado, del almacén.
Si el almacén está vacío (sin videojuegos) o si no hay ningún videojuego en dicha posición dentro del almacén, visualizará en consola el mensaje:

```
Error al eliminar: almacén vacío o
                   posición no indica videojuego en el almacén.
```

Si la eliminación se ha realizado con éxito, visualizará en consola el mensaje:

```
Se ha eliminado el videojuego del almacén con éxito.
```

3) Consultar un videojuego, por posición, del almacén.

Leerá por teclado la posición del videojuego a consultar.
Realizará la consulta del videojuego, indicado mediante la posición leída desde teclado, del almacén.
Si el almacén contiene un videojuego en dicha posición, visualizará en consola el videojuego encontrado, indicando un resumen con los valores de todos los atributos de dicho videojuego.
En caso contrario, visualizará en consola el mensaje:

```
No se ha encontrado ningún videojuego en la posición del almacén.
```

4) Consultar todos los videojuegos del almacén.

Realizará la consulta de todos los videojuegos del almacén.
Si no hay videojuegos en el almacén, visualizará en consola el mensaje:

```
El almacén está vacío.
```

En caso contrario, visualizará en consola el listado de todos los videojuegos contenidos en el almacén, indicando para cada videojuego, su posición dentro del almacén y el resumen con los valores de todos los atributos de dicho videojuego. Por ejemplo:

```
(0) Videojuego[Código=1, Título=BotW, Precio=49.95, Cantidad=10]
(1) Videojuego[Código=2, Título=GTA5, Precio=19.95, Cantidad=10]
(2) Videojuego[Código=3, Título=FH4, Precio=29.95, Cantidad=10]
```

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 4). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 4.
```

HERENCIA DE CLASES Y POLIMORFISMO

ACTIVIDAD 2x01

Una empresa de autobuses tiene distintos tipos de empleados. Por una lado, están los conductores de los autobuses, y por otro lado, el personal de oficinas (oficinistas) y los limpiadores, que se encargan de oficinas y autobuses, respectivamente.

- De cada **empleado**, se desea saber su nif, nombre completo, fecha de nacimiento, dirección completa, un número de teléfono y un correo electrónico.
- De cada **oficinista**, se quiere guardar la titulación que poseen, así como el turno en el que trabajan (diurno o vespertino).
- De cada **conductor**, se quiere saber el número de carnets de conducir de los que disponen y el año de obtención de su CAP.
- De cada **limpiador**, se debe saber si limpian oficinas, autobuses o las dos cosas. En el futuro se plantea la posibilidad de que también limpien garajes.

Puede ocurrir que en un momento dado, algún empleado cambie de dirección o de teléfono. El personal de oficinas puede obtener una nueva titulación y también puede cambiar el turno en el que trabaja. Por su parte, los conductores pueden obtener un nuevo carnet y también renovar su CAP. Además, los limpiadores pueden variar lo que limpian.

Algunos aspectos que interesa conocer de esta empresa son los siguientes:

- El número máximo de carnets que se puede sacar un conductor son 15. Podría darse el caso de que este dato cambiase en el futuro.
- De momento no hay turno nocturno en oficinas, pero hay previsión de implantarlo.

Diseña una jerarquía de clases relativa a los empleados de esta empresa. Codifica las clases **`Empleado`**, **`Conductor`**, **`Oficinista`** y **`Limpiador`**, incluyendo en cada una los atributos necesarios, un constructor, los métodos necesarios y el método sobrescrito **`toString`**.

Codifica una clase **`Actividad_2x01`** que incluya un programa principal **`main`** que realice lo siguiente:

- Instanciará varios objetos de cada clase diseñada (por ejemplo: dos empleados, dos conductores, tres oficinistas y tres limpiadores) y los almacenará en un vector de empleados con 10 posiciones.
- Visualizará en consola todos los empleados del vector, indicando, para cada uno de ellos, un mensaje indicando el tipo con el que está instanciado y un resumen con los valores de todos sus atributos.

Un ejemplo de ejecución del programa podría ser:

```
El empleado 0 es un empleado genérico.
NIF=00000000A, Nombre=Juan, FechaNacimiento=12/12/1950,
Dirección=C/. Naranja, Teléfono=777888999, Correo=juan@gmail.com

El empleado 1 es un empleado genérico.
NIF=11111111B, Nombre=Ana, FechaNacimiento=06/02/1985,
Dirección=C/. Violeta, Teléfono=777666555, Correo=ana@gmail.com
```

```
El empleado 2 es un oficinista.
Oficinista [NIF=22222222C, Nombre=Jorge, FechaNacimiento=30/07/1988,
    Dirección=C/. Azul, Teléfono=777000444, Correo=jorge@gmail.com
    Titulación=Administración de Empresas, Turno=diurno]

El empleado 3 es un oficinista.
Oficinista [NIF=33333333D, Nombre=Isabel, FechaNacimiento=12/11/1999,
    Dirección=C/. Amarillo, Teléfono=777222333, Correo=isabel@gmail.com
    Titulación=Derecho, Turno=vespertino]

El empleado 4 es un oficinista.
Oficinista [NIF=44444444E, Nombre=Elena, FechaNacimiento=15/06/1998,
    Dirección=C/. Verde, Teléfono=777111666, Correo=elena@gmail.com
    Titulación=Economía, Turno=nocturno]

El empleado 5 es un conductor.
Conductor [NIF=55555555F, Nombre=Mateo, FechaNacimiento=25/09/1978,
    Dirección=C/. Rojo, Teléfono=666111444, Correo=mateo@gmail.com
    NúmeroCarnets=5, AñoCAP=2000]

El empleado 6 es un conductor.
Conductor [NIF=66666666G, Nombre=Teresa, FechaNacimiento=18/03/1959,
    Dirección=C/. Blanco, Teléfono=666222333, Correo=teresa@gmail.com
    NúmeroCarnets=15, AñoCAP=1970]

El empleado 7 es un limpiador.
Limpiador [NIF=77777777H, Nombre=Manuel, FechaNacimiento=09/05/1983,
    Dirección=C/. Negro, Teléfono=666000999, Correo=manuel@gmail.com
    LimpiaOficinas=true, LimpiaAutobuses=false, LimpiaGarajes=false]

El empleado 8 es un limpiador.
Limpiador [NIF=88888888I, Nombre=María, FechaNacimiento=13/07/2003,
    Dirección=C/. Gris, Teléfono=666111888, Correo=maria@gmail.com
    LimpiaOficinas=false, LimpiaAutobuses=true, LimpiaGarajes=false]

El empleado 9 es un limpiador.
Limpiador [NIF=99999999J, Nombre=Pedro, FechaNacimiento=21/08/2007,
    Dirección=C/. Marrón, Teléfono=666222777, Correo=pedro@gmail.com
    LimpiaOficinas=false, LimpiaAutobuses=false, LimpiaGarajes=true]
```

ACTIVIDAD 2x02

Una empresa se dedica a vender bebidas. De cada **bebida** se desea guardar un código identificativo (que se generará automáticamente de forma incremental) y el nombre. Se desea disponer de una operación que permita calcular el precio de una bebida en función de su tipo.

La empresa vende dos tipos de bebidas: refrescos y bebidas alcohólicas.

- De cada **refresco** se desea almacenar si tiene gas o no, y además la cantidad de azúcar.
  El nombre de un refresco será uno de los siguientes: `limonada`, `cola`, `gaseosa`, `té helado`, `granizado` o `ponche`. La cantidad de azúcar de un refresco estará comprendida entre 4 y 35 gramos (por cada 100 mililitros).
  El precio de un refresco se obtendrá según la siguiente fórmula:
  `precio = 1 + (cantidad azúcar / 10)`
- De cada **bebida alcohólica** únicamente se almacena su graduación de alcohol. A su vez, hay dos tipos de bebidas alcohólicas: bebidas fermentadas y bebidas destiladas.
- De cada **bebida fermentada** no se necesita guardar más datos.
  El nombre de una bebida fermentada será uno de los siguientes: `sidra`, `cerveza`, `vino`, `sake` o `hidromiel`. La graduación de alcohol de una bebida fermentada estará comprendida entre 3,5 y 15 grados (en volumen).
  El precio de una bebida fermentada se obtendrá según la siguiente fórmula:
  `precio = 5 + (graduación alcohol / 10)`
- De cada **bebida destilada** no se necesita guardar más datos.
  El nombre de una bebida destilada será uno de los siguientes: `anís`, `pacharán`, `ginebra`, `cognac`, `ron`, `vodka`, `whisky` o `tequila`. La graduación de alcohol de una bebida destilada estará comprendida entre 15 y 45 grados (en volumen).
  El precio de una bebida destilada se obtendrá según la siguiente fórmula:
  `precio = 12 + (graduación alcohol / 10)`

Diseña una jerarquía de clases relativa a las bebidas vendidas por esta empresa. Codifica las clases **`Bebida`**, **`Refresco`**, **`Alcoholica`**, **`Fermentada`** y **`Destilada`**, incluyendo en cada una los atributos necesarios, un constructor, los métodos necesarios y el método sobrescrito **`toString`**. No se deberá poder instanciar objetos con las clases `Bebida` y `Alcoholica`.

Codifica una clase **`Inventario`** que permita gestionar un contenedor de bebidas. Este inventario contendrá lo siguiente:

- Un vector de bebidas.
- Un índice que tendrá una doble función: indicará el número de bebidas almacenadas que están instanciadas dentro del vector y marcará además la primera posición libre del vector.
- Un constructor que recibirá como parámetro la capacidad del vector (el número de elementos que tiene el vector). Al instanciar un objeto de esta clase, el vector se creará con la capacidad dada y el índice del inventario será 0.
- El método sobrescrito **`toString`** que devolverá una cadena de texto con la información de todas las bebidas del inventario. Para cada bebida, se indicará su posición dentro del vector de bebidas y un resumen con los valores de todos los atributos de dicha bebida.
- Los métodos de objeto necesarios, según las opciones de menú indicadas en el programa principal.

Codifica una clase **`Actividad_2x02`** que incluya un programa principal **`main`**. Este programa utilizará un inventario de bebidas (con capacidad para 20 bebidas) y gestionará el siguiente menú de opciones:

0) Salir del programa.

1) Rellenar el inventario con bebidas generadas de forma aleatoria.

Leerá por teclado el número de bebidas a insertar en el inventario.
Para cada bebida a insertar, determinará de forma aleatoria el tipo de bebida (refresco, bebida fermentada o bebida destilada):

- Si la bebida a insertar es un refresco, intentará realizar la inserción de un refresco, instanciado con datos generados de forma aleatoria, en el inventario.
- Si la bebida a insertar es una bebida fermentada, intentará realizar la inserción de una bebida fermentada, instanciada con datos generados de forma aleatoria, en el inventario.
- Si la bebida a insertar es una bebida destilada, intentará realizar la inserción de una bebida destilada, instanciada con datos generados de forma aleatoria, en el inventario.

Visualizará en consola un mensaje indicando el número de bebidas de cada tipo insertadas en el inventario.
Por ejemplo:

```
¿Número de Bebidas? 10
Se han insertado 3 refrescos en el inventario.
Se han insertado 2 bebidas fermentadas en el inventario.
Se han insertado 5 bebidas destiladas en el inventario.
```

2) Consultar todas las bebidas del inventario.

Realizará la consulta de todas las bebidas del inventario.
Si no hay bebidas en el inventario, visualizará en consola el mensaje:

```
El inventario está vacío.
```

En caso contrario:

- Visualizará en consola el listado de todas las bebidas contenidas en el inventario, indicando para cada bebida, su posición dentro del inventario y un resumen con los valores de todos los atributos de dicha bebida.
- Visualizará en consola el número de bebidas consultadas del inventario.

Por ejemplo:

```
(0) Refresco [Código=1, Nombre=ponche,
              Gaseoso=true, CantidadAzúcar=34,5]
(1) Destilada [Código=2, Nombre=cognac, GraduaciónAlcohol=16,2]
(2) Fermentada [Código=3, Nombre=cerveza, GraduaciónAlcohol=6,7]
(3) Fermentada [Código=4, Nombre=sake, GraduaciónAlcohol=11,5]
(4) Refresco [Código=5, Nombre=granizado,
              Gaseoso=true, CantidadAzúcar=21,8]
(5) Destilada [Código=6, Nombre=cognac, GraduaciónAlcohol=28,0]
(6) Destilada [Código=7, Nombre=tequila, GraduaciónAlcohol=32,1]
(7) Refresco [Código=8, Nombre=cola,
              Gaseoso=false, CantidadAzúcar=29,8]
(8) Destilada [Código=9, Nombre=ginebra, GraduaciónAlcohol=41,6]
(9) Destilada [Código=10, Nombre=cognac, GraduaciónAlcohol=28,0]
Se han consultado 10 bebidas del inventario.
```

3) Consultar los refrescos del inventario.

Realizará la consulta de los refrescos del inventario.
Visualizará en consola el listado de los refrescos contenidos en el inventario, indicando para cada refresco, su posición dentro del inventario y un resumen con los valores de todos los atributos de dicho refresco.
Visualizará en consola el número de refrescos consultados del inventario.
Por ejemplo:

```
(0) Refresco [Código=1, Nombre=ponche,
              Gaseoso=true, CantidadAzúcar=34,5]
(4) Refresco [Código=5, Nombre=granizado,
              Gaseoso=true, CantidadAzúcar=21,8]
(7) Refresco [Código=8, Nombre=cola,
              Gaseoso=false, CantidadAzúcar=29,8]
Se han consultado 3 refrescos del inventario.
```

4) Consultar las bebidas fermentadas del inventario.

Realizará la consulta de las bebidas fermentadas del inventario.
Visualizará en consola el listado de las bebidas fermentadas contenidas en el inventario, indicando para cada bebida fermentada, su posición dentro del inventario y un resumen con los valores de todos los atributos de dicha bebida fermentada.
Visualizará en consola el número de bebidas fermentadas consultadas del inventario.
Por ejemplo:

```
(2) Fermentada [Código=3, Nombre=cerveza, GraduaciónAlcohol=6,7]
(3) Fermentada [Código=4, Nombre=sake, GraduaciónAlcohol=11,5]
Se han consultado 2 bebidas fermentadas del inventario.
```

5) Consultar las bebidas destiladas del inventario.

Realizará la consulta de las bebidas destiladas del inventario.
Visualizará en consola el listado de las bebidas destiladas contenidas en el inventario, indicando para cada bebida destilada, su posición dentro del inventario y un resumen con los valores de todos los atributos de dicha bebida destilada.
Visualizará en consola el número de bebidas destiladas consultadas del inventario.
Por ejemplo:

```
(1) Destilada [Código=2, Nombre=cognac, GraduaciónAlcohol=16,2]
(5) Destilada [Código=6, Nombre=cognac, GraduaciónAlcohol=28,0]
(6) Destilada [Código=7, Nombre=tequila, GraduaciónAlcohol=32,1]
(8) Destilada [Código=9, Nombre=ginebra, GraduaciónAlcohol=41,6]
(9) Destilada [Código=10, Nombre=cognac, GraduaciónAlcohol=28,0]
Se han consultado 5 bebidas destiladas del inventario.
```

6) Consultar las bebidas con precio comprendido entre dos límites.

Leerá por teclado un precio mínimo y un precio máximo.
Realizará la consulta de las bebidas cuyo precio está comprendido entre el precio mínimo y el precio máximo (con ambos límites incluidos).
Visualizará en consola el listado de las bebidas consultadas del inventario, indicando para cada bebida, su posición dentro del inventario, un resumen con los valores de todos los atributos de dicha bebida y su precio calculado.
Visualizará en consola el número de bebidas consultadas del inventario.

Por ejemplo:

```
¿Precio Mínimo? 3.50
¿Precio Máximo? 14.20
(0) Refresco [Código=1, Nombre=ponche,
              Gaseoso=true, CantidadAzúcar=34,5]
    Precio=4,45
(1) Destilada [Código=2, Nombre=cognac, GraduaciónAlcohol=16,2]
    Precio=13,62
(2) Fermentada [Código=3, Nombre=cerveza, GraduaciónAlcohol=6,7]
    Precio=5,67
(3) Fermentada [Código=4, Nombre=sake, GraduaciónAlcohol=11,5]
    Precio=6,15
(7) Refresco [Código=8, Nombre=cola,
              Gaseoso=false, CantidadAzúcar=29,8]
    Precio=3,98
Se han consultado 5 bebidas del inventario.
```

7) **Calcular el número de refrescos gaseosos del inventario.**

Calculará y visualizará en consola el número de refrescos gaseosos contenidos en el inventario.
Por ejemplo:

```
Número de Refrescos Gaseosos: 2
```

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 7). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 7.
```

IMPLEMENTACIÓN DE INTERFACES Y POLIMORFISMO

ACTIVIDAD 3x01

Codifica una interfaz **`PrecioModificable`** que contenga los siguientes métodos sin implementación:

- `double` **`calcularPrecioConIVA`**`()`
  Calculará y devolverá el precio con IVA.
- `double` **`calcularPrecioOferta`**`(double descuento)`
  Calculará y devolverá el precio de oferta, tras aplicar un descuento.
- `double` **`calcularPrecioPorCantidad`**`(int cantidad)`
  Calculará y devolverá el precio por cantidad, tras comprar una gran cantidad de unidades.

Codifica una clase **`Libro`** que incluya lo siguiente:

- Los atributos necesarios para guardar la información relativa al título, año de publicación, escritor y precio base de cada libro.
- Un constructor que reciba varios parámetros y que inicialice los atributos del objeto con ellos.
- El método sobrescrito **`toString`** que devolverá una cadena de texto con el estado del objeto (nombre de la clase y un resumen con los valores de todos los atributos del objeto).

Además, esta clase **`Libro`** deberá implementar la interfaz **`PrecioModificable`**:

- Los libros tendrán un IVA del 4%.
- Se aplicarán las siguientes reglas para calcular el precio por cantidad de los libros:
  - Si se compran de 10 a 99 libros, se aplicará un descuento del 4%.
  - Si se compran de 100 a 999 libros, se aplicará un descuento del 8%.
  - Si se compran 1000 o más libros, se aplicará un descuento del 16%.

Codifica una clase **`Pelicula`** que incluya lo siguiente:

- Los atributos necesarios para guardar la información relativa al título, año de estreno, director y precio base de cada película.
- Un constructor que reciba varios parámetros y que inicialice los atributos del objeto con ellos.
- El método sobrescrito **`toString`** que devolverá una cadena de texto con el estado del objeto (nombre de la clase y un resumen con los valores de todos los atributos del objeto).

Además, esta clase **`Pelicula`** deberá implementar la interfaz **`PrecioModificable`**:

- Las películas tendrán un IVA del 10%.
- Se aplicarán las siguientes reglas para calcular el precio por cantidad de las películas:
  - Si se compran de 5 a 49 películas, se aplicará un descuento del 3%.
  - Si se compran de 50 a 499 películas, se aplicará un descuento del 6%.
  - Si se compran 500 o más películas, se aplicará un descuento del 12%.

Codifica una clase **Actividad_3x01** que incluya un programa principal **main** que realice lo siguiente:

- Instanciará varios objetos de cada clase diseñada (por ejemplo: tres libros y tres películas) y los almacenará en un vector de precios modificables con 6 posiciones.
- Visualizará en consola todos los precios modificables del vector, indicando, para cada uno de ellos, un mensaje indicando el tipo con el que está instanciado, un resumen con los valores de todos sus atributos, el precio con IVA, el precio de oferta con un descuento del 10% y el precio por cantidad al comprar 100 unidades.

Un ejemplo de ejecución del programa podría ser:

```
El precio modificable 0 es un libro.
Libro [Título=Los Tres Mosqueteros, AñoPublicación=1844,
       Escritor=Alexandre Dumas, Precio=29,50]
Precio con IVA:                      30,68
Precio de Oferta (descuento 10%):    27,61
Precio por Cantidad (100 unidades): 28,23

El precio modificable 1 es un libro.
Libro [Título=El Señor de los Anillos, AñoPublicación=1955,
       Escritor=J.R.R. Tolkien, Precio=59,95]
Precio con IVA:                      62,35
Precio de Oferta (descuento 10%):    56,11
Precio por Cantidad (100 unidades): 57,36

El precio modificable 2 es un libro.
Libro [Título=Dune, AñoPublicación=1965,
       Escritor=Frank Herbert, Precio=37,25]
Precio con IVA:                      38,74
Precio de Oferta (descuento 10%):    34,87
Precio por Cantidad (100 unidades): 35,64

El precio modificable 3 es una película.
Película [Título=Ben-Hur, AñoEstreno=1959,
          Director=William Wyler, Precio=18,99]
Precio con IVA:                      20,89
Precio de Oferta (descuento 10%):    18,80
Precio por Cantidad (100 unidades): 19,64

El precio modificable 4 es una película.
Película [Título=Titanic, AñoEstreno=1997,
          Director=James Cameron, Precio=23,50]
Precio con IVA:                      25,85
Precio de Oferta (descuento 10%):    23,27
Precio por Cantidad (100 unidades): 24,30

El precio modificable 5 es una película.
Película [Título=Los Vengadores, AñoEstreno=2012,
          Director=Joss Whedon, Precio=21,75]
Precio con IVA:                      23,93
Precio de Oferta (descuento 10%):    21,53
Precio por Cantidad (100 unidades): 22,49
```

ACTIVIDAD 3x02

Una aplicación gráfica dibuja figuras en dos dimensiones en una pantalla. De cada **figura 2D** se desea guardar un código identificativo (que se generará automáticamente de forma incremental), el grosor del borde y el color. Se desea disponer de dos operaciones que permitan calcular el perímetro y el área de una figura 2D en función de su tipo.

La aplicación gráfica dibuja tres tipos de figuras 2D: triángulos equiláteros, rectángulos y círculos.

- De cada **triángulo equilátero** se desea almacenar el lado.
  El perímetro de un triángulo equilátero se obtendrá según la siguiente fórmula:
  `perímetro = 3 * lado`
  El área de un triángulo equilátero se obtendrá según la siguiente fórmula:
  `área = lado * lado * (sqrt(3) / 4)`
- De cada **rectángulo** se desea almacenar la base y la altura.
  El perímetro de un rectángulo se obtendrá según la siguiente fórmula:
  `perímetro = (2 * base) + (2 * altura)`
  El área de un rectángulo se obtendrá según la siguiente fórmula:
  `área = base * altura`
- De cada **círculo** se desea almacenar el radio.
  El perímetro de un círculo se obtendrá según la siguiente fórmula:
  `perímetro = 2 * PI * radio`
  El área de un círculo se obtendrá según la siguiente fórmula:
  `área = PI * radio * radio`

Diseña una jerarquía de clases relativa a las figuras 2D dibujadas por esta aplicación gráfica. Codifica las clases **`Figura2D`**, **`TrianguloEquilatero`**, **`Rectangulo`** y **`Circulo`**, incluyendo en cada una los atributos necesarios, un constructor, los métodos necesarios y el método sobrescrito **`toString`**. No se deberá poder instanciar objetos con la clase `Figura2D`.

Codifica una interfaz **`Operativa`** que contenga los siguientes métodos sin implementación:

- `boolean` **`insertar`**`(Object objeto)`
  Insertará un objeto en un contenedor. Devolverá verdadero si la inserción se ha realizado con éxito o falso en caso contrario.
- `boolean` **`actualizar`**`(int posicion, Object objeto)`
  Actualizará un objeto, por posición, de un contenedor con los datos de otro objeto. Devolverá verdadero si la actualización se ha realizado con éxito o falso en caso contrario.
- `boolean` **`eliminar`**`(int posicion)`
  Eliminará un objeto, por posición, de un contenedor. Devolverá verdadero si la eliminación se ha realizado con éxito o falso en caso contrario.
- `Object` **`consultar`**`(int posicion)`
  Consultará un objeto, por posición, de un contenedor. Devolverá el objeto que se encuentra en la posición del contenedor si la consulta se ha realizado con éxito. Devolverá `null` si no hay ningún objeto en la posición del contenedor.

Codifica una clase **`Pantalla`** que permita gestionar un contenedor de figuras 2D y que implemente la interfaz **`Operativa`**. Esta pantalla contendrá lo siguiente:

- Un vector de figuras 2D.
- Un índice que tendrá una doble función: indicará el número de figuras 2D almacenadas que están instanciadas dentro del vector y marcará además la primera posición libre del vector.
- Un constructor que recibirá como parámetro la capacidad del vector (el número de elementos que tiene el vector). Al instanciar un objeto de esta clase, el vector se creará con la capacidad dada y el índice del inventario será 0.
- El método sobrescrito **`toString`** que devolverá una cadena de texto con la información de todas las figuras 2D de la pantalla. Para cada figura 2D, se indicará su posición dentro del vector de figuras 2D y un resumen con los valores de todos los atributos de dicha figura 2D.
- Los métodos de objeto de la interfaz **`Operativa`** sobrescritos con la implementación adecuada para esta clase **`Pantalla`**.
- Otros métodos de objeto necesarios, según las opciones de menú indicadas en el programa principal.

Codifica una clase **`Actividad_3x02`** que incluya un programa principal **`main`**. Este programa utilizará una pantalla de figuras 2D (con capacidad para 15 figuras 2D) y gestionará el siguiente menú de opciones:

0) Salir del programa.

1) Insertar una figura 2D en la pantalla.

Leerá por teclado el tipo (triángulo equilátero, rectángulo o círculo) de la figura 2D a insertar.
Leerá por teclado los datos de la figura 2D a insertar:
- Si la figura a insertar es un triángulo equilátero, leerá por teclado el grosor del borde, el color y el lado.
- Si la figura a insertar es un rectángulo, leerá por teclado el grosor del borde, el color, la base y la altura.
- Si la figura a insertar es un círculo, leerá por teclado el grosor del borde, el color y el radio.

Intentará realizar la inserción de la figura 2D, creada a partir de los datos leídos por teclado, en la pantalla.
Si la pantalla está llena (con 15 figuras 2D), visualizará en consola el mensaje:

```
Error al insertar: pantalla llena.
```

Si la inserción se ha realizado con éxito, visualizará en consola el mensaje:

```
Se ha insertado la figura 2D en la pantalla con éxito.
```

2) Actualizar una figura 2D, por posición, de la pantalla.

Leerá por teclado la posición de la figura 2D a actualizar.
Leerá por teclado el tipo (triángulo equilátero, rectángulo o círculo) de la figura 2D a actualizar.
Leerá por teclado los datos de la figura 2D a actualizar:
- Si la figura a actualizar es un triángulo equilátero, leerá por teclado el grosor del borde, el color y el lado.
- Si la figura a actualizar es un rectángulo, leerá por teclado el grosor del borde, el color, la base y la altura.
- Si la figura a actualizar es un círculo, leerá por teclado el grosor del borde, el color y el radio.

Intentará realizar la actualización de la figura 2D, indicada mediante la posición leída desde teclado, de la pantalla con los nuevos datos.

Si la pantalla está vacía (sin figuras 2D) o si no hay ninguna figura 2D en dicha posición dentro de la pantalla, visualizará en consola el mensaje:

```
Error al actualizar: pantalla vacía o
                     posición no indica figura 2D en la pantalla o
                     tipo de figura 2D no coincidente.
```

Si la actualización se ha realizado con éxito, visualizará en consola el mensaje:

```
Se ha actualizado la figura 2D de la pantalla con éxito.
```

3) Eliminar una figura 2D, por posición, de la pantalla.

Leerá por teclado la posición de la figura 2D a eliminar.
Intentará realizar la eliminación de la figura 2D, indicada mediante la posición leída desde teclado, de la pantalla.
Si la pantalla está vacía (sin figuras 2D) o si no hay ninguna figura 2D en dicha posición dentro de la pantalla, visualizará en consola el mensaje:

```
Error al eliminar: pantalla vacía o
                   posición no indica figura 2D en la pantalla.
```

Si la eliminación se ha realizado con éxito, visualizará en consola el mensaje:

```
Se ha eliminado la figura 2D de la pantalla con éxito.
```

4) Consultar una figura 2D, por posición, de la pantalla.

Leerá por teclado la posición de la figura 2D a consultar.
Realizará la consulta de la figura 2D, indicada mediante la posición leída desde teclado, de la pantalla.
Si la pantalla contiene una figura 2D en dicha posición, visualizará en consola la figura 2D encontrada, indicando un resumen con los valores de todos los atributos de dicha figura 2D.
En caso contrario, visualizará en consola el mensaje:

```
No se ha encontrado ninguna figura 2D en la posición de la pantalla.
```

5) Consultar todas las figuras 2D de la pantalla.

Realizará la consulta de todas las figuras 2D de la pantalla.
Si no hay figuras 2D en la pantalla, visualizará en consola el mensaje:

```
La pantalla está vacía.
```

En caso contrario:

- Visualizará en consola el listado de todas las figuras 2D contenidas en la pantalla, indicando para cada figura 2D, su posición dentro de la pantalla, un resumen con los valores de todos los atributos de dicha figura 2D, incluyendo además su perímetro y su área.
- Visualizará en consola el número de figuras 2D consultadas de la pantalla.

6) Ordenar la pantalla por grosor de borde de forma ascendente.

Ordenará todas las figuras 2D de la pantalla por el grosor del borde de forma ascendente.
Consultará todas las figuras 2D de la pantalla y visualizará en consola un listado con estas figuras 2D de la pantalla ordenadas con este criterio de ordenación.

7) Ordenar la pantalla por color de forma ascendente.

Ordenará todas las figuras 2D de la pantalla por el color de forma ascendente.
Consultará todas las figuras 2D de la pantalla y visualizará en consola un listado con estas figuras 2D de la pantalla ordenadas con este criterio de ordenación.

8) **Ordenar la pantalla por perímetro de forma descendente.**

   Ordenará todas las figuras 2D de la pantalla por el perímetro de forma descendente.
   Consultará todas las figuras 2D de la pantalla y visualizará en consola un listado con estas figuras 2D de la pantalla ordenadas con este criterio de ordenación.

9) **Ordenar la pantalla por área de forma descendente.**

   Ordenará todas las figuras 2D de la pantalla por el área de forma descendente.
   Consultará todas las figuras 2D de la pantalla y visualizará en consola un listado con estas figuras 2D de la pantalla ordenadas con este criterio de ordenación.

Este programa principal deberá validar que la opción de menú elegida sea válida (comprendida entre 0 y 9). Si no lo es, visualizará en consola el mensaje:

```
La opción de menú debe estar comprendida entre 0 y 9.
```