

Ejercicios: Interfaces

1 – Ordenar viajes

Para la clase viaje del ejercicio:

<https://drive.google.com/file/d/1w5y9KiTO1wtDTiOodrXspKk3YH0b-OJ4/view?usp=sharing>

Implementar la interfaz IComparable para permitir al usuario el ingreso indefinido de Viajes en un Array para poder ordenarlos de menor a mayor por precio y luego mostrar su orden inverso. Se debe usar el método Sort y Reverse de la clase Array.

2- Buscar en un listado de Componentes

Para la clase viaje del ejercicio:

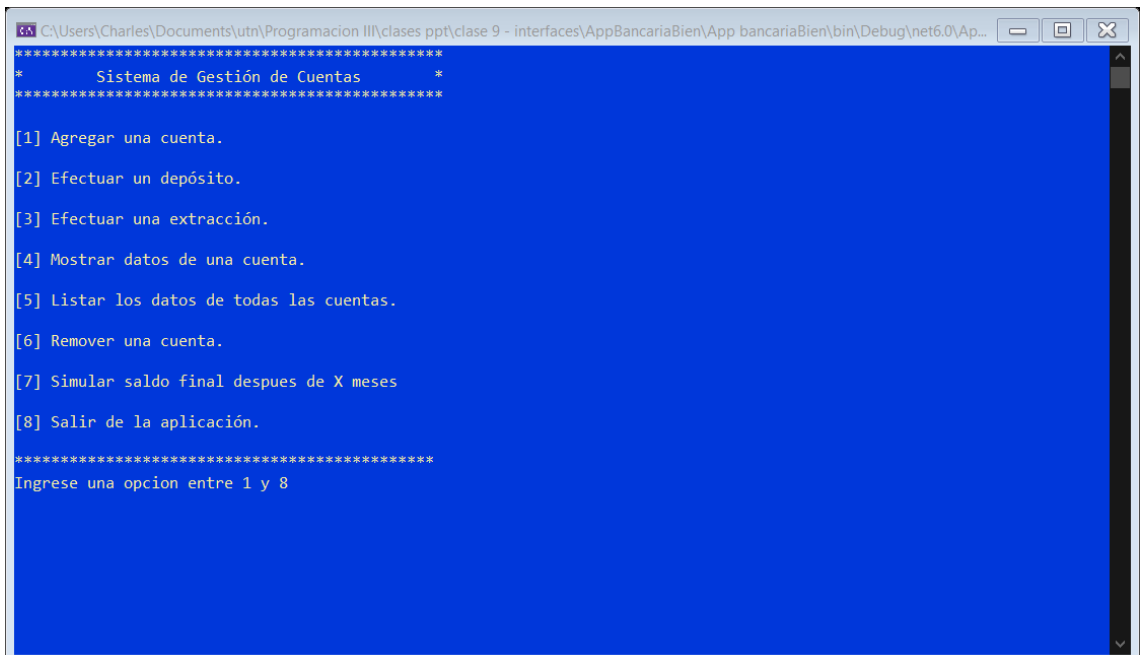
<https://drive.google.com/file/d/1HICB8LYwg9YSBz09BiB5kblQMWO0tLIS/view?usp=sharing>

Implementar la interfaz IEquatable para permitir al usuario el ingreso indefinido de componentes. Luego, permitir al usuario el ingreso de un numero de serie y que muestre los datos completos del mismo y permita modificar los datos a través de un menú.

3 – App bancaria 3.0

Para la app bancaria que se realizo en la clase anterior realizar las siguientes modificaciones:

1. Agregar un atributo estático interesMensual del tipo flotante con sus correspondientes getters y setters. El atributo interés mensual es la tasa de interés (es un porcentaje) que el banco le dará por mes al cliente por tener su dinero depositado.
2. Al inicio del programa, antes de ingresar el menú, se deberá ingresar la tasa de interés para setearla.
3. Agregar un elemento al menú que ingresando una cantidad de meses devuelva cual será el monto final según la tasa de interés mensual ingresada. No lo debe modificar. El menú se deberá ver así:



```
C:\Users\Charles\Documents\utn\Programacion III\clases ppt\clase 9 - interfaces\AppBancariaBien\App bancariaBien\bin\Debug\net6.0\Ap...
*****
*      Sistema de Gestión de Cuentas      *
*****

[1] Agregar una cuenta.
[2] Efectuar un depósito.
[3] Efectuar una extracción.
[4] Mostrar datos de una cuenta.
[5] Listar los datos de todas las cuentas.
[6] Remover una cuenta.
[7] Simular saldo final despues de X meses
[8] Salir de la aplicación.

*****
Ingrese una opción entre 1 y 8
```

4. Implementar la interfaz IEquatable para que se pueda buscar usando el método IndexOf.

5. Implementar la interfaz IComparable para que al mostrar los datos de todas las cuentas se muestren en orden de mayor a menor saldo (o mayor o menor CBU, a elección del alumno).

4 – Venta de autopartes

Un cliente, requiere un sistema modelado e implementado observando los principios de la POO para registrar y administrar las ventas de autopartes de un comercio de venta de repuestos del automotor. Los datos pertinentes a dicho objeto Autoparte son:

- GANANCIA: Número real (punto flotante de precisión simple) – Valor único compartido para todas las instancias del objeto. Representa el porcentaje de incremento sobre el costo que se desea establecer como margen de ganancia. Ejemplo: 25,75%.
- Código de Pieza: Número entero del intervalo [0 a 999.999.999]. Ejemplo: 733.615.993.
- Descripción: Cadena de caracteres descriptiva de la Autoparte. Ejemplo: “Correa de Distribución”.
- Costo: Número real (punto flotante de precisión simple) que representa el valor de reposición del artículo. Ejemplo: \$189,50.

La Clase Autoparte además de contar con esos atributos debe contar con los métodos necesarios como getters, setters, constructores y un método darPrecio(cuotas) que recibiendo por valor como argumento la cantidad de cuotas de pago en que se abonará (número entero corto positivo), devuelva como número real (punto flotante de precisión simple), el valor total a abonar, conforme adicionar una penalización del 10% por cada cuota, aplicada sobre el monto original (interés simple – Si cuotas es igual a 1, el recargo es 0%). Tener en cuenta que para obtener el precio final se debe sumar la ganancia que se busca obtener.

Además, se deben implementar las interfaces IComparable de la siguiente manera: Buscamos comparar dos objetos de la clase Autoparte según el costo. También, se debe usar la interfaz IComparable buscando el Equals según el código de pieza.

Para la clase ejecutora se pide:

1. Al inicio del programa, se deberá pedir al usuario el ingreso de la ganancia que se busca obtener por cada autoparte.
2. Ingresar una cantidad indeterminada de autopartes impidiendo que se ingresen dos Autopartes iguales. El ingreso de autopartes deberá finalizar con el código 0.
3. Una vez que se ingresan se deberá pedir el ingreso de una cantidad indeterminada de códigos de autopartes para buscarlo. En caso de que se encuentre se deberá pedir la cantidad de cuotas para mostrar su precio final. El ingreso de códigos deberá finalizar con el código 0.
4. Una vez finalizado el ingreso se deberán mostrar todas las autopartes ingresadas ordenadas de mayor a menor por precio y luego de menor a mayor.

A elección del alumno: Se debe separar el programa en por lo menos cuatro clases:

1. Una clase interfaz, que maneje el ingreso y salida de datos del programa.
2. Una clase controladora, que maneje la lista de autopartes como también la búsqueda y el agregado de autopartes a la misma o su ordenamiento.
3. Una clase Autoparte con los atributos y métodos correspondientes.
4. Una clase ejecutora (o Program, por defecto) que pueda llamar a las distintas clases del programa según sea necesario.

