using System;

namespace AlgoritmosFinalOK

{

class Program

{

struct DatosPelicula

{

public string nombre; // Nombre de la película

public double costoAdulto; // Costo del boleto de adulto

public double costoMenores; // Costo del boleto de menor

public int boletosVendidosMenores; // Cantidad de boletos vendidos de esta película a menores

public int boletosVendidosAdulto; // Cantidad de boletos vendidos de esta película a adultos

public double dineroDescontado; // Cantidad de dinero descontado

}

static void Main(string[] args)

{

int n; // Cantidad de películas

EscribeYLeeInt("Cantidad de películas:", "ERROR: TU CANTIDAD NO PUEDE SER NEGATIVA Y DEBE SER UN NÚMERO.", out n, 0, int.MaxValue);

DatosPelicula[] datosPeliculas = new DatosPelicula[n];

// Función que maneja el llenado de los datos de las películas en cartelera

Console.WriteLine("\n\n\n ==== FASE 1. LLENADO DE DATOS DE LA CARTELERA ==== ");

DatosCartelera(datosPeliculas);

// Función que maneja el llenado de los datos de las películas en cuanto a ventas

Console.WriteLine("\n\n\n ==== FASE 2. LLENADO DE DATOS DE VENTAS ==== ");

Ventas(datosPeliculas);

// Función que maneja el reporte de los resultados de las dos fases anteriores

Console.WriteLine("\n\n\n ==== FASE 3. REPORTE DE INFORMACIÓN ==== ");

MuestraResultados(datosPeliculas);

Console.WriteLine("\n\n\n ==== ¡HASTA LUEGO! ==== ");

Console.ReadLine();

}

// ===== FASE 1 =====

// Procesa la primera fase (datos cartelera) del programa

static void DatosCartelera(DatosPelicula[] lista)

{

for (int i = 0; i < lista.Length; i++)

{

// Escribe el nombre de la película en la estructura y la valida

lista[i].nombre = ValidacionNombrePelicula("Escribe el nombre de la película " + (i + 1) + ":", lista, i);

// Llena el costo del boleto de adulto y valida

EscribeYLeeDouble("Escribe el costo del boleto de adulto para la película " + (i + 1) + ":", "ERROR: TU COSTO NO PUEDE SER NEGATIVO Y DEBE SER UN NÚMERO.", out lista[i].costoAdulto, 0, double.MaxValue);

// Llena el costo del boleto de menor y valida

EscribeYLeeDouble("Escribe el costo del boleto de menores para la película " + (i + 1) + ":", "ERROR: TU COSTO NO PUEDE SER NEGATIVO Y DEBE SER UN NÚMERO.", out lista[i].costoMenores, 0, double.MaxValue);

// Asigna los demás valores a cero ya que se irán agregando aditivamente

lista[i].boletosVendidosAdulto = 0;

lista[i].boletosVendidosMenores = 0;

lista[i].dineroDescontado = 0;

}

}

// UTILIDADES FASE 1

static string ValidacionNombrePelicula(string mensaje, DatosPelicula[] peliculas, int indice)

{

string nombre; // El nombre propuesto por el usuario

bool nombreValido; // ¿El nombre es válido?

do

{

// Por defecto es válido, y se invalida si cumple con ciertos criterios

nombreValido = true;

// Escribe el mensaje que pregunta al usuario el nombre

Console.WriteLine(mensaje);

// Asigna a nombre lo que escribe el usuario en consola

nombre = Console.ReadLine();

// si el nombre sin contar espacios está vacío...

if (nombre.Replace(" ", "") == "")

{

// ...lanza un error y vuelve a hacer el do-while loop desde el inicio

Console.WriteLine("ERROR: El nombre de la película no puede quedar en blanco");

nombreValido = false;

continue;

}

// para cada película que ya llenamos...

for (int i = 0; i < indice; i++)

{

// ...si el nombre propuesto sin contar mayúsculas/minúsculas ni espacios se repite...

if (nombre.ToLower().Replace(" ", "") == peliculas[i].nombre.ToLower().Replace(" ", ""))

{

// ...lanza un error y vuelve a hacer el do-while loop desde el inicio

Console.WriteLine("ERROR: El nombre de la película no puede repetirse exactamente, de ser una versión distinta se debe diferenciar en el nombre");

nombreValido = false;

continue;

}

}

} while(!nombreValido);

return nombre;

}

// ===== FASE 2 =====

// Procesa la segunda fase (ventas) del programa

static void Ventas(DatosPelicula[] listas)

{

int clientes; // La cantidad de clientes

EscribeYLeeInt("¿Cuantos clientes son?",

"ERROR: Debe de haber por lo menos un cliente y debe ser un número",

out clientes, 1, int.MaxValue);

// Por cada cliente, ejecutar este código

for (int j = 0; j < clientes; j++)

{

int boletosAdulto, boletosMenores, indicePelicula;

double ventaAdulto, ventaMenor, descontado;

// Obtiene el índice de la película según su nombre

indicePelicula = IndicePeliculaPorNombre(listas, "Escribe el nombre de la película que va ver el cliente [" + (j+1) + "]");

// Pregunta, valida y almacena la cantidad de boletos de adulto del cliente

EscribeYLeeInt("¿Cuantos boletos de adulto va a comprar el cliente [" + (j + 1) + "]?",

"ERROR: La cantidad de boletos no puede ser negativa y debe ser un número",

out boletosAdulto, 0, int.MaxValue);

// Pregunta, valida y almacena la cantidad de boletos de menor del cliente

EscribeYLeeInt("¿Cuantos boletos de menor va a comprar el cliente [" + (j + 1) + "]?",

"ERROR: La cantidad de boletos no puede ser negativa y debe ser un número",

out boletosMenores, 0, int.MaxValue);

// Conserva la cantidad de venta de adulto y de menor mediante la multiplicación de los boletos por su costo

ventaAdulto = listas[indicePelicula].costoAdulto \* boletosAdulto;

ventaMenor = listas[indicePelicula].costoMenores \* boletosMenores;

// Verifica si hay descuento y guárdalo

if (boletosAdulto + boletosMenores >= 3 && boletosMenores >= 1)

descontado = ventaMenor \* 0.30;

else

descontado = 0;

// Añade el descuento y la cantidad de boletos vendidos a la información de la película

listas[indicePelicula].dineroDescontado += descontado;

listas[indicePelicula].boletosVendidosAdulto += boletosAdulto;

listas[indicePelicula].boletosVendidosMenores += boletosMenores;

// Escribe un desglose del cliente para depurar el proceso y para tener mayor transparencia

Console.WriteLine("\nDESGLOSE CLIENTE: ");

Console.WriteLine("Boletos menores: " + boletosMenores);

Console.WriteLine("Boletos adultos: " + boletosAdulto);

Console.WriteLine("Venta menores (sin contar descuento): $" + ventaMenor);

Console.WriteLine("Venta adultos: $" + ventaAdulto);

Console.WriteLine("Venta total: $" + (ventaAdulto + ventaMenor));

Console.WriteLine("Descontado: $" + descontado);

Console.WriteLine("Película: " + listas[indicePelicula].nombre);

Console.WriteLine("\n\n");

}

}

// UTILIDADES FASE 2

// Encuentra el índice de la película en el arreglo mediante su nombre

static int IndicePeliculaPorNombre(DatosPelicula[] peliculas, string mensaje)

{

string nombre;

int indiceEncontrado = -1; // -1 es el valor asignado si no se encuentra nada, guardará el índice de la película en nuestro arreglo

bool encontrado = false;

do // mientras que no se encuentre un índice dado un nombre, do-while

{

// Escribe el mensaje que pregunta al usuario por el nombre

Console.WriteLine("\n" + mensaje);

// Almacena la propuesta de nombre

nombre = Console.ReadLine();

// Para cada película...

for (int i = 0; i < peliculas.Length; i++)

{

// ... si el nombre propuesto por el usuario sin contar mayúsculas/minúsculas ni espacios se encuentra en el arreglo en el índice i...

if (peliculas[i].nombre.ToLower().Replace(" ", "") == nombre.ToLower().Replace(" ", ""))

{

//...asigna el índice de esa película a indiceEncontrado, haciendo que ya no sea -1

indiceEncontrado = i;

}

}

if (indiceEncontrado == -1)

Console.WriteLine("No se encontró esa película"); // Si es -1, no se ha encontrado película

else

encontrado = true; // Si no es -1, se encontró la película

} while(!encontrado);

return indiceEncontrado; // Regresa el índice de la película

}

// ===== FASE 3 =====

// Procesa la fase 3 (reporte de resultados) del programa

static void MuestraResultados(DatosPelicula[] tabla)

{

// Es el menú que muestra las opciones de reporte en un string verbatim para facilitar su lectura en el código.

string menuReporte = @"ESCRIBE EL NÚMERO DE LA OPCIÓN DESEADA

1) Mostrar ventas en de todas las peliculas como tabla

2) Mostrar datos de una sóla película

3) Mostrar datos de las peliculas que contengan en su título un texto

4) Mostrar película con más boletos vendidos

5) Mostrar película con menos boletos vendidos

6) Mostrar película con mayores recaudaciones (tras descuentos)

7) Mostrar película con menores recaudaciones (tras descuentos)

8) Mostrar cantidad de boletos vendidos a menores

9) Mostrar cantidad de boletos vendidos a adultos

10) Mostrar la cantidad de boletos vendidos

11) Mostrar cobro total (sin descuentos), descuentos totales y ganancia neta

12) Mostrar película con la mayor cantidad de dinero descontado

13) Mostrar película con la menor cantidad de dinero descontado

14) Salir";

int opcion; // Almacena la opción seleccionada por el usuario

do

{

// Valida y almacena la opción que seleccionó el usuario

EscribeYLeeInt(menuReporte, "ERROR: Esa no es una opción válida", out opcion, 1, 14);

// Un espacio en blanco para que se vea bonito ^-^

Console.WriteLine();

// Decide qué hacer según la opción

switch (opcion)

{

case 1: // 1) Mostrar ventas en de todas las peliculas como tabla

EscribeTablaPeliculas(tabla);

break;

case 2: // 2) Mostrar datos de una sóla película

BuscaPeliculaNombreExacto(tabla);

break;

case 3: // 3) Mostrar datos de las peliculas que contengan en su título un texto

BuscaPeliculaQueContenganTexto(tabla);

break;

case 4: // 4) Mostrar película con más boletos vendidos

EscribeDatosPelicula(PeliculaMasVendida(tabla), true);

break;

case 5: // 5) Mostrar película con menos boletos vendidos

EscribeDatosPelicula(PeliculaMenosVendida(tabla), true);

break;

case 6: // 6) Mostrar película con mayores recaudaciones (tras descuentos) (recaudaciones = boletosAdulto\*costoBoletoAdulto + boletosMenor \* costoBoletoMenor - descuentos)

EscribeDatosPelicula(PeliculaMasRecaudaciones(tabla), true);

break;

case 7: // 7) Mostrar película con menores recaudaciones (tras descuentos)

EscribeDatosPelicula(PeliculaMenosRecaudaciones(tabla), true);

break;

case 8: // 8) Mostrar cantidad de boletos vendidos a menores

Console.WriteLine("Se vendieron " + BoletosMenores(tabla) + " boletos a menores");

break;

case 9: // 9) Mostrar cantidad de boletos vendidos a adultos

Console.WriteLine("Se vendieron " + BoletosAdultos(tabla) + " boletos a adultos");

break;

case 10: // 10) Mostrar la cantidad de boletos vendidos

Console.WriteLine("Se vendieron " + BoletosTotales(tabla) + " boletos en total");

break;

case 11: // 11) Mostrar cobro total (sin descuentos), descuentos totales y ganancia neta

CobroDescuentosGanancia(tabla);

break;

case 12: // 12) Mostrar película con la mayor cantidad de dinero descontado

EscribeDatosPelicula(MayorDescuento(tabla), true);

break;

case 13: // 13) Mostrar película con la menor cantidad de dinero descontado

EscribeDatosPelicula(MenorDescuento(tabla), true);

break;

case 14: // 14) Salir

break;

default: // Default, continúa

break;

}

Console.WriteLine();

} while(opcion != 14);

}

// FUNCIONES FASE 3

//case 1: Mostrar ventas en de todas las peliculas como tabla

static void EscribeTablaPeliculas(DatosPelicula[] peliculas)

{

for (int i = 0; i < peliculas.Length; i++)

{

// Por cada película, escribe su fila y el encabezado si es la primera

EscribeDatosPelicula(peliculas[i], i == 0);

}

}

//case 2: Mostrar datos de una sóla película

static void BuscaPeliculaNombreExacto(DatosPelicula[] peliculas)

{

string nombre;

DatosPelicula pelicula = peliculas[0]; // Asignado a cero para evitar error de valor no asignado

bool encontrado = false;

Console.WriteLine("\nEscribe el nombre de la película");

// Almacena el nombre de la película

nombre = Console.ReadLine();

// Por cada película...

for(int i = 0; i < peliculas.Length; i++)

{

//...si su nombre sin contar espacios ni mayúsculas/minúsculas es igual al sugerido por el usuario...

if(peliculas[i].nombre.ToLower().Replace(" ", "") == nombre.ToLower().Replace(" ", ""))

{

//...asigna a pelicula la información de esa película y marca que fue encontrada una película bajo ese nombre

pelicula = peliculas[i];

encontrado = true;

}

}

if(!encontrado)

Console.WriteLine("No se encontró esa película"); // si no se marca como encontrada la película, no hay

else

EscribeDatosPelicula(pelicula, true); // de lo contrario, escribe una tabla que contenga sólo su columna de información

}

//case 3: Mostrar datos de las peliculas que contengan en su título un texto

static void BuscaPeliculaQueContenganTexto(DatosPelicula[] peliculas)

{

string texto;

// Un arreglo que contiene las películas encontradas. Su tamaño es de la cantidad de películas ya que es el peor caso donde toda película contiene ese texto

DatosPelicula[] peliculasEncontradas = new DatosPelicula[peliculas.Length];

// ¿Cuántos se encontraron? Nos sirve saberlo para saber hasta dónde leer el arreglo de la línea de arriba y donde meter la siguiente búsqueda

int encontrados = 0;

Console.WriteLine("\nEscribe el texto que debe contener el nombre de la película");

// Guarda el texto sugerido por el usuario y quita espacios y convierte a minúsculas para ignorar mayúsculas/minúsculas a la hora de evaluar

texto = Console.ReadLine().ToLower().Replace(" ", "");

// Para cada película...

for (int i = 0; i < peliculas.Length; i++)

{

//...si la película contiene el texto (ignorando espacios y mayúsculas/minúsculas)...

if (peliculas[i].nombre.ToLower().Replace(" ", "").Contains(texto))

{

//...añade al arreglo e incrementa la cantidad de encontrados

peliculasEncontradas[encontrados] = peliculas[i];

encontrados++;

}

}

if (encontrados == 0)

Console.WriteLine("No se encontró película que contenga ese texto"); // si no se encontró, lanza un mensaje que lo indique

else

{

//Si se encontro...

//... para cada encontrado...

for(int i = 0; i < encontrados; i++)

{

//...escribe sus datos como una columna de una tabla y ponle encabezado si es el primer encontrado

EscribeDatosPelicula(peliculasEncontradas[i], i == 0);

}

}

}

//case 4: Mostrar película con más boletos vendidos

static DatosPelicula PeliculaMasVendida(DatosPelicula[] lista)

{

double max = -1;

int total = 0;

DatosPelicula peliculaMasVendida = lista[0];

// Para cada película...

for (int i = 0; i < lista.Length; i++)

{

// ...encuentra el total, compáralo con el máximo, y si lo es, designa la película más vendida a la actual

total = lista[i].boletosVendidosAdulto + lista[i].boletosVendidosMenores;

if (total > max)

{

max = total;

peliculaMasVendida = lista[i];

}

}

return peliculaMasVendida;

}

//case 5: Mostrar película con menos boletos vendidos

static DatosPelicula PeliculaMenosVendida(DatosPelicula[] lista)

{

double min = double.MaxValue;

int total = 0;

DatosPelicula peliculaMenosVendida = lista[0];

// Para cada película...

for (int i = 0; i < lista.Length; i++)

{

// ...encuentra el total, compáralo con el mínimo, y si lo es, designa la película menos vendida a la actual

total = lista[i].boletosVendidosAdulto + lista[i].boletosVendidosMenores;

if (total < min)

{

min = total;

peliculaMenosVendida = lista[i];

}

}

return peliculaMenosVendida;

}

//case 6: Mostrar película con mayores recaudaciones (tras descuentos)

static DatosPelicula PeliculaMasRecaudaciones(DatosPelicula[] lista)

{

double recaudaciones, max = -1;

DatosPelicula peliculaMasRecaudaciones = lista[0];

// Para cada película...

for (int i = 0; i < lista.Length; i++)

{

// ...encuentra la recaudación, compáralo con el máximo y, de ser mayor, asígna la película actual como la mayor

recaudaciones = (lista[i].boletosVendidosAdulto \* lista[i].costoAdulto) + (lista[i].boletosVendidosMenores \* lista[i].costoMenores);

if (recaudaciones > max)

{

max = recaudaciones;

peliculaMasRecaudaciones = lista[i];

}

}

return peliculaMasRecaudaciones;

}

//case 7: Mostrar película con menores recaudaciones (tras descuentos)

static DatosPelicula PeliculaMenosRecaudaciones(DatosPelicula[] lista)

{

double recaudaciones, min = double.MaxValue;

DatosPelicula peliculaMenosRecaudaciones = lista[0];

// Para cada película...

for (int i = 0; i < lista.Length; i++)

{

// ...encuentra la recaudación, compáralo con el mínimo y, de ser menor, asígna la película actual como la menor

recaudaciones = (lista[i].boletosVendidosAdulto \* lista[i].costoAdulto) + (lista[i].boletosVendidosMenores \* lista[i].costoMenores);

if (recaudaciones < min)

{

min = recaudaciones;

peliculaMenosRecaudaciones = lista[i];

}

}

return peliculaMenosRecaudaciones;

}

//case 8: Mostrar cantidad de boletos vendidos a menores

static int BoletosMenores(DatosPelicula[] lista)

{

int total = 0;

for (int i = 0; i < lista.Length; i++)

{

// Para cada película, añade su cantidad de boletos vendidos a menores al total.

total = total + lista[i].boletosVendidosMenores;

}

return total;

}

//case 9: Mostrar cantidad de boletos vendidos a adultos

static int BoletosAdultos(DatosPelicula[] lista)

{

int total = 0;

for (int i = 0; i < lista.Length; i++)

{

// Para cada película, añade su cantidad de boletos vendidos a adultos al total.

total = total + lista[i].boletosVendidosAdulto;

}

return total;

}

//case 10: Mostrar la cantidad de boletos vendidos

static int BoletosTotales(DatosPelicula[] lista)

{

// La cantidad de boletos totales es la suma de boletos de adultos y de menores

return BoletosAdultos(lista) + BoletosMenores(lista);

}

//case 11: Mostrar cobro total (sin descuentos), descuentos totales y ganancia neta

static void CobroDescuentosGanancia(DatosPelicula[] peliculas)

{

double cobroTotal = 0, descuentosTotales = 0, gananciaNeta;

// Para cada película...

for(int i = 0; i < peliculas.Length; i++)

{

//...al cobro total añádele el cobro de esa película...

cobroTotal += (peliculas[i].costoAdulto \* peliculas[i].boletosVendidosAdulto)

+ (peliculas[i].costoMenores \* peliculas[i].boletosVendidosMenores);

//...y a los descuentos, añádele los descuentos de esa película

descuentosTotales += peliculas[i].dineroDescontado;

}

// La ganancia neta es el cobro total menos los descuentos totales

gananciaNeta = cobroTotal - descuentosTotales;

// Escribe la información obtenida

Console.WriteLine("Cobro total (no tomando descuentos en cuenta): $" + cobroTotal);

Console.WriteLine("Dinero aplicado en descuentos: $" + descuentosTotales);

Console.WriteLine("Ganancia neta: $" + gananciaNeta);

}

//case 12: Mostrar película con la mayor cantidad de dinero descontado

static DatosPelicula MayorDescuento(DatosPelicula[] peliculas)

{

DatosPelicula mayorDescuento = peliculas[0];

// Para cada película...

for (int i = 1; i < peliculas.Length; i++)

{

//... si el descuento es mayor al máximo, asigna la película con mayor descuento a esta

if (peliculas[i].dineroDescontado > mayorDescuento.dineroDescontado)

mayorDescuento = peliculas[i];

}

return mayorDescuento;

}

//case 13: Mostrar película con la menor cantidad de dinero descontado

static DatosPelicula MenorDescuento(DatosPelicula[] peliculas)

{

DatosPelicula menorDescuento = peliculas[0];

// Para cada película...

for (int i = 1; i < peliculas.Length; i++)

{

//... si el descuento es menor al mínimo, asigna la película con menor descuento a esta

if (peliculas[i].dineroDescontado < menorDescuento.dineroDescontado)

menorDescuento = peliculas[i];

}

return menorDescuento;

}

// UTILIDADES FASE 3

// Escribe la información de una película como una columna con encabezado opcional

static void EscribeDatosPelicula(DatosPelicula pelicula, bool encabezado)

{

// Si queremos encabezado, escribe el encabezado formateado con los siguientes espacios y títulos

if (encabezado)

Console.WriteLine(FormatearFilaTabla("Nombre de la película", "Costo Adulto", "Costo Menor", "Boletos Adulto", "Boletos Menores", "Cobro Adulto", "Cobro Menor", "Cobro Total", "Descuento", "Ganancia Total"

, 40, 20, 20, 20, 20, 20, 20, 20, 20, 20));

// Escribe los datos de la película formateado con los siguientes espacios

Console.WriteLine(ResultadosPelicula(pelicula, 40, 20, 20, 20, 20, 20, 20, 20, 20, 20));

}

// Formatea los datos de la película en una fila con espacios especificados

static string ResultadosPelicula(DatosPelicula pelicula, int charsNombre, int charsCostoAdulto, int charsCostoMenores, int charsBoletosAdulto, int charsBoletosMenores, int charsCobroAdulto, int charsCobroMenor, int charsCobroTotal, int charsDescuento, int charsTotal)

{

// Encuentra el string de cada cosa que queremos que aparezca en la tabla si es información

string nombre = pelicula.nombre;

string costoAdulto = "$" + pelicula.costoAdulto;

string costoMenores = "$" + pelicula.costoMenores;

string boletosAdulto = "" + pelicula.boletosVendidosAdulto;

string boletosMenores = "" + pelicula.boletosVendidosMenores;

string recaudadoAdulto = "$" + (pelicula.costoAdulto \* pelicula.boletosVendidosAdulto);

string recaudadoMenor = "$" + (pelicula.costoMenores \* pelicula.boletosVendidosMenores);

string recaudadoTotal = "$" + ((pelicula.costoAdulto \* pelicula.boletosVendidosAdulto) + (pelicula.costoMenores \* pelicula.boletosVendidosMenores));

string descuento = "$" + pelicula.dineroDescontado;

string total = "$" + ((pelicula.costoAdulto \* pelicula.boletosVendidosAdulto) + (pelicula.costoMenores \* pelicula.boletosVendidosMenores) - pelicula.dineroDescontado);

// Formatea los strings anteriores y lo regresa como uno sólo

return FormatearFilaTabla(nombre, costoAdulto, costoMenores, boletosAdulto, boletosMenores, recaudadoAdulto, recaudadoMenor, recaudadoTotal, descuento, total,

charsNombre, charsCostoAdulto, charsCostoMenores, charsBoletosAdulto, charsBoletosMenores, charsCobroAdulto, charsCobroMenor, charsCobroTotal, charsDescuento, charsTotal);

}

// Formatea una fila de la tabla

static string FormatearFilaTabla(string nombre, string costoAdulto, string costoMenores, string boletosAdulto, string boletosMenores, string cobroAdulto, string cobroMenor, string cobroTotal, string descuento, string total, int charsNombre, int charsCostoAdulto, int charsCostoMenores, int charsBoletosAdulto, int charsBoletosMenores, int charsCobroAdulto, int charsCobroMenor, int charsCobroTotal, int charsDescuento, int charsTotal)

{

// el textoVacio se añade al final de cada campo para que si necesitamos más caracteres de los que mide el campo para formatear la tabla, se utilicen espacios

string textoVacio = " ";

nombre += textoVacio;

costoAdulto += textoVacio;

costoMenores += textoVacio;

boletosAdulto += textoVacio;

boletosMenores += textoVacio;

cobroAdulto += textoVacio;

cobroMenor += textoVacio;

cobroTotal += textoVacio;

descuento += textoVacio;

total += textoVacio;

// Se obtiene un string del tamaño deseado al recortar el string anterior al tamaño que se especifica en su parámentro

nombre = nombre.Substring(0, charsNombre);

costoAdulto = costoAdulto.Substring(0, charsCostoAdulto);

costoMenores = costoMenores.Substring(0, charsCostoMenores);

boletosAdulto = boletosAdulto.Substring(0, charsBoletosAdulto);

boletosMenores = boletosMenores.Substring(0, charsBoletosMenores);

cobroAdulto = cobroAdulto.Substring(0, charsCobroAdulto);

cobroMenor = cobroMenor.Substring(0, charsCobroMenor);

cobroTotal = cobroTotal.Substring(0, charsCobroTotal);

descuento = descuento.Substring(0, charsDescuento);

total = total.Substring(0, charsTotal);

// Regresa todos los strings formateados juntos

return nombre + costoAdulto + costoMenores + boletosAdulto + boletosMenores + cobroAdulto + cobroMenor + cobroTotal + descuento + total;

}

// ===== UTILIDADES =====

// Escribe un mensaje que pide la entrada de un int al usuario, lee el valor, lo valida, lanza un error si no es válido.

static void EscribeYLeeInt(string textoPedir, string textoError, out int resultado, int resultadoMayorIgualQue, int resultadoMenorIgualQue)

{

bool continuar = true;

do

{

// Pide al usuario el valor

Console.WriteLine(textoPedir);

// Revisa si se pudo parsear el Console.ReadLine y pasarse efectivamente al resultado. También verifica si el valor de éste se encuentra en el rango especificado.

if (int.TryParse(Console.ReadLine(), out resultado) && resultado >= resultadoMayorIgualQue && resultado <= resultadoMenorIgualQue)

continuar = false; // Si obtiene resultado y es válido, puede terminar el do-while loop

else

Console.WriteLine("\t" + textoError); // De no ser válido, lanza el error

} while (continuar);

}

// Escribe un mensaje que pide la entrada de un double al usuario, lee el valor, lo valida, lanza un error si no es válido.

static void EscribeYLeeDouble(string textoPedir, string textoError, out double resultado, double resultadoMayorIgualQue, double resultadoMenorIgualQue)

{

bool continuar = true;

do

{

// Pide al usuario el valor

Console.WriteLine(textoPedir);

// Revisa si se pudo parsear el Console.ReadLine y pasarse efectivamente al resultado. También verifica si el valor de éste se encuentra en el rango especificado.

if (double.TryParse(Console.ReadLine(), out resultado) && resultado >= resultadoMayorIgualQue && resultado <= resultadoMenorIgualQue)

continuar = false; // Si obtiene resultado y es válido, puede terminar el do-while loop

else

Console.WriteLine("\t" + textoError); // De no ser válido, lanza el error

} while (continuar);

}

}

}