Unidad 2

DIPLOMATURA EN PROGRAMACION .NET

Ejercicios

Capítulo 3

Control del Programa y Vectores

Ejercicio 1

En este ejercicio se utilizará un ciclo y sentencias de control de flujo

Crear una aplicación que realiza 50 ciclos e imprima en una línea separada cada valor de la variable de control del ciclo. Se deberá imprimir además de este valor para cada múltiplo de 3 “mTres”, para cada múltiplo de 5 “mCinco” y para cada múltipo de 7 “mSiete”

Por ejemplo:

1
2
3 mTres
4
5 mCinco
6 mTres
7 mSiete
8
9 mTres
10 mCinco
11
12 mTres
13
14 mSiete
15 mTres mCinco
16

etc...

Tareas:

1. Abrir la solución en el directorio Capitulo3
2. Crear un proyecto llamarlo ejercicio1
3. Programar dentro del método Main un ciclo `for` o `For` (C# o VB) y donde la variable de iteración se llame numero.
4. Dentro del ciclo `for` o `For` (C# o VB), realizar salidas por consola utilizando el método Console.Write cuando sea conveniente. Este método a diferencia de Console.WriteLine no imprime un carácter de nueva línea.
5. Dentro del ciclo `for` o `For` (C# o VB) utilizar una sentencia `if` o `If` (C# o VB) para determinar cuando deben imprimirse y cuando no, “mTres”, “mCinco” y “mSiete”. Sugerencia: el operador % o `Mod` (C# o VB) calcula el resto de la división entera.
6. Por último dentro del mismo ciclo `for` o `For` (C# o VB), imprimir una nueva línea.

Ejercicio 2

En este ejercicio se adquirirá experiencia en declarar, crear y manipular vectores de una dimensión de tipos primitivos

Realizar los siguientes pasos:

1. Abrir la solución en el directorio Capitulo3 si no la tiene abierta ya
2. Crear un proyecto llamarlo ejercicio2
3. En el método Main() declarar dos variables de referencia a vector de enteros llamadas vector1 y vector2.
4. Inicializar el vector1 con un bloque de inicialización (valores entre llaves) que contenga los ocho primeros números primos: 2, 3, 5, 7, 11, 17 y 19.
5. Mostrar por pantalla el contenido de vector1. Utilizar el método ImprimeVector que se muestra a continuación:

C#

```
public void ImprimeVector(int[] vector)
{
    Console.Write('<');
    for (int i = 0; i < vector.Length; i++)
    {
        // Imprimir un elemento
        Console.Write(vector[i]);
        // Imprimir una coma para delimitar si no es el último elemento
        if ((i + 1) < vector.Length)
        {
            Console.Write(", ");
        }
    }
    Console.Write('>');
    Console.WriteLine();
}
```

VB

```
Public Sub ImprimeVector(vector() As Integer)
    Console.Write("<")
    For i = 0 To vector.Length - 1
        ' Imprimir un elemento
        Console.Write(vector(i).ToString)
        ' Imprimir una coma para delimitar si no es el último elemento
        If ((i + 1) < vector.Length) Then
            Console.Write(", ")
        End If
    Next
    Console.Write(">")
    Console.WriteLine()
End Sub
```

En el caso de usar C#, Crear una instancia de la clase Program e invocar al método con notación de punto. En el caso de VB, incluir el método en el módulo e invocarlo directamente

6. Compilar y ejecutar el programa
7. Asignar la variable vector1 a la variable vector2. Modificar cada elemento de vector2 para que su contenido sea igual a su número de índice (Ej: vector2[0]=0 o vector2(0)=0 – C# o VB), teniendo en cuenta que consta de siete elementos.
8. Imprimir vector1 y verificar que pasó con su contenido.

Ejercicio 3

En este ejercicio se adquirirá experiencia en declarar, crear y manipular vectores de dos dimensiones de tipos primitivos

Tareas

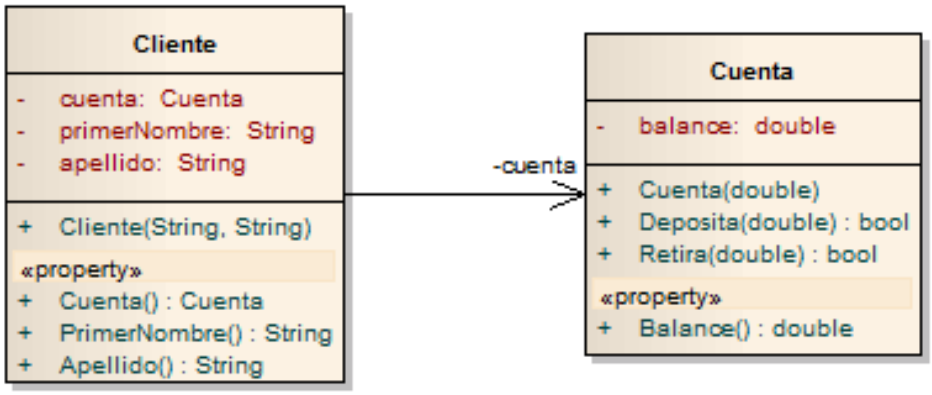
Realizar los siguientes pasos:

1. Abrir la solución en el directorio Capitulo3 si no la tiene abierta ya
2. Crear un proyecto llamarlo ejercicio3
3. Crear una clase de aplicación (con el método main) y llamarla PruebaDeVectores. Declarar una variable llamada matriz de tipo entero (int[][] – un vector de vectores de enteros)
4. Asignar elementos a la matriz de manera de producir la siguiente salida:

```
matriz[0] es <>
matriz[1] es <0>
matriz[2] es <0, 2>
matriz[3] es <0, 3, 6>
matriz[4] es <0, 4, 8, 12>
```

Ejercicio 4

Se debe modificar los métodos deposita y retira para retornar un valor booleano que especifique si la transacción fue satisfactoria



Tareas

1. Abrir la solución del directorio Capítulo3 si no la tiene abierta ya y trabajar en el proyecto ejercicio4

2. Modificar la clase cuenta para que condicione los métodos deposita y retira

3. Modificar el método deposita para retornar `true` (significa que el depósito fue satisfactorio)
4. Modificar el método retira para que verifique que la cantidad que se intenta retirar no sea mayor al balance actual. Si la cantidad es menor que el balance, descontar dicha cantidad y retornar `true`, caso contrario no descontar nada de balance y devolver `false`
5. Compilar y ejecutar el Main de la clase PruebaOperacionesBancarias para verificar que la salida sea la siguiente:

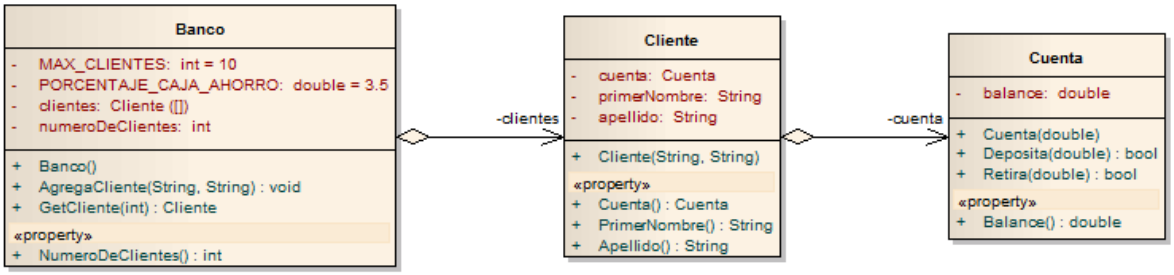
```
Creando al cliente Juan Perez.
Creando una cuenta con 500.00 de balance.
Retira 150.00 : True
Deposita 22.50 : True
Retira 47.62 : True
La cuenta tiene un balance de 347,38
Cliente [Perez, Juan] tiene un balance de 347,38
```


Ejercicio 5

En este ejercicio se implementará la multiplicidad que existe en la asociación entre un banco y sus clientes

Tareas

1. Abrir el espacio de trabajo del directorio Capítulo 3\Ejercicio 5. Los errores que figuran se deben a las modificaciones que deberán realizarse.
2. Crear la clase Banco dentro del espacio de nobres operacionesBancarias. Cuando se cree un objeto del tipo banco se creará una relación entre él y los objetos del tipo Cliente. Esta se implementa como una agregación en un vector de objetos del tipo Cliente.



3. Agregar dos atributos a la clase Banco: clientes (un vector de objetos del tipo Cliente) y numeroDeClientes (un entero con el que se sabe cual es el próximo cliente en el vector)
4. Agregar un constructor público que inicialice el vector cliente con una cantidad de elementos superior a 5.
5. Añadir el método AgregaClientes(), en él se deberá construir los objetos del tipo Cliente con sus respectivos parámetros (nombre y apellido), colocarlo dentro del vector cliente e incrementar en uno el atributo numeroDeClientes
6. Agregar la propiedad de sólo lectura NumeroDeClientes el cual retorna el atributo numeroDeClientes
7. Agregar el método GetCliente() el cual retornará el objeto del tipo Cliente según el índice que se pase como parámetro.
8. Compilar y ejecutar el programa.
9. Verificar que la salida sea:

```
Cliente [0] es Juan Pérez
Cliente [1] es Pedro García
Cliente [2] es Oscar Toma
Cliente [3] es María Soley
```