Universidad Tecnológica
del Suroeste de Guanajuato

Ingeniería en Tecnologías de la Información y Comunicación

Nombre de la alumna:
María Guadalupe Botello Chacón
Gloria Estefany Chacón González
América Janet Méndez Negrete

Nombre del Maestro:
Ariana Gómez Contreras

Materia:
Programación de aplicaciones

Fecha de entrega:
22/08/2019

Las buenas prácticas de programación son un conjunto formal o informal de reglas, donde pueden ser opcionales u obligatorias, que se adoptan con el fin general de mejorar la calidad del software.

En particular, se pueden obtener beneficios específicos tales como: facilitar el proceso de desarrollo para el programador.

Las buenas prácticas existen en las diversas etapas del proceso de desarrollo de software. En este documento nos enfocamos en las buenas prácticas de programación para el lenguaje C porque se está trabajando con el MVC (Modelo Vista Controlador).

Con este modelo, como su nombre nos indica separamos el modelo, la vista y el controlador, esta separación nos ayudara a que no se fuerce el sistema al cargar todos los métodos para terminar utilizando solo uno de ellos, lo cual da la seguridad de que el sistema o aplicación trabajen a un nivel adecuado sin forzar la máquina donde se está ejecutando y se puedan lograr los resultados esperados.

La aplicación que se creó utilizando el MVC fue la siguiente:

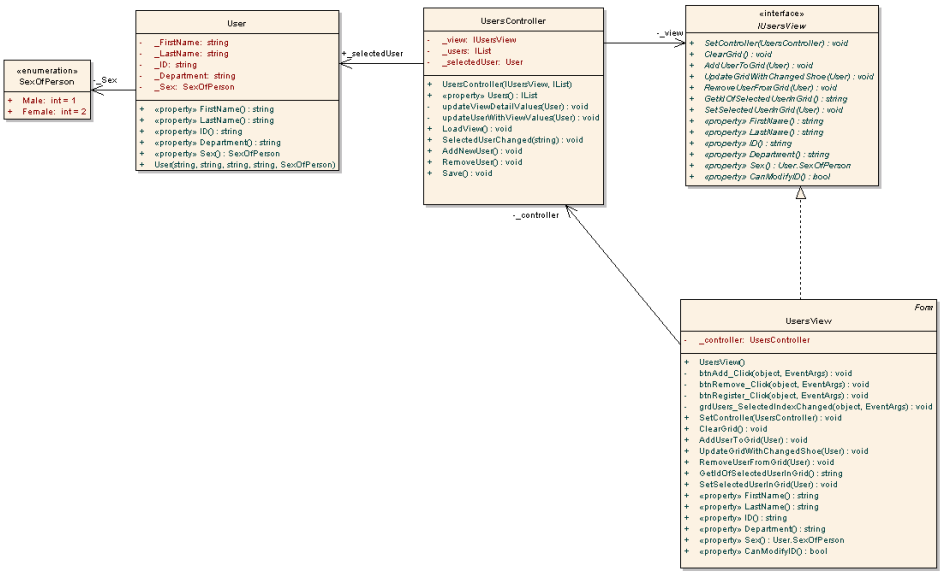

Figura 1. Diagrama de Clase.

En la parte del controlador se crearon los métodos que son necesarios para alojar la información y que son con los que se va a trabajar, esto para no hacer la carga pesada a la vista y esta haga lo menos posible, este trabajo el controlador lo hace por medio de una Interfaz.

```
7
8   namespace MVC.Controller
9   {
10          public interface IUsersView
11          {
12              void SetController(UsersController controller);
13              void ClearGrid();
14              void AddUserToGrid(User user);
15              void UpdateGridWithChangedUser(User user);
16              void RemoveUserFromGrid(User user);
17              string GetIdOfSelectedUserInGrid();
18              void SetSelectedUserInGrid(User user);
19
20              string FirstName { get; set; }
21              string LastName { get; set; }
22              string ID { get; set; }
23              string Department { get; set; }
24              User.SexOfPerson Sex { get; set; }
25              bool CanModifyID { set; }
26          }
27
28      public class UsersController
29      {
30          IUsersView _view;
31          IList _users;
32          User _selectedUser;
33
34          public UsersController(IUsersView view, IList users)
35          {
36              _view = view;
37              users = users;
```

La vista carga la lista de usuarios, esto lo hace implementando el controlador, que en este caso es una interfaz.

```
1   using System;
2   using System.Collections.Generic;
3   using System.Collections;
4   using System.Text;
5   using MVC.Controller;
6   using MVC.Model;
7
8   namespace MVC.View
9   {
10      public partial class UsersView : Form, IUsersView
11      {
12          public void SetController(UsersController controller)
13          {
14              _controller = controller;
15          }
```

Y cargamos la vista con la lista de elementos que se quieren mostrar

```
public void AddUserToGrid(User usr)
{
    ListViewItem parent;
    parent = this.grdUsers.Items.Add(usr.ID);
    parent.SubItems.Add(usr.FirstName);
    parent.SubItems.Add(usr.LastName);
    parent.SubItems.Add(usr.Department);
    parent.SubItems.Add(Enum.GetName(typeof(User.SexOfPerson), usr.Sex));
}

public void UpdateGridWithChangedUser(User usr)
{
    ListViewItem rowToUpdate = null;

    foreach (ListViewItem row in this.grdUsers.Items)
    {
        if (row.Text == usr.ID)
        {
            rowToUpdate = row;
        }
    }

    if (rowToUpdate != null)
    {
        rowToUpdate.Text = usr.ID;
        rowToUpdate.SubItems[1].Text = usr.FirstName;
        rowToUpdate.SubItems[2].Text = usr.LastName;
```

El modelo nos ayuda a obtener los datos, que además son privados y las propiedades para el código del cliente. Además, contiene el estado de memoria y este hace operaciones no tan pesadas pero que le dan valor al proceso.

```
5   namespace MVC.Model
6   {
7       public class User
8       {
9           public enum SexOfPerson
10          {
11              Male = 1,
12              Female = 2
13          }
14
15          private string _FirstName;
16          public string FirstName
17          {
18              get { return _FirstName; }
19              set
20              {
21                  if (value.Length > 50)
22                      Console.WriteLine("Error! FirstName must be less than 51 characters!");
23                  else
24                      _FirstName = value;
25              }
26          }
27
28          private string _LastName;
29          public string LastName
30          {
31              get { return _LastName; }
32              set
33              {
34                  if (value.Length > 50)
```

Y la parte del cliente, donde se implementa el modelo vista controlador, pues ya que desde aquí hacemos el llamado de los componentes.

```
using System.Collections;
using MVC.Model;
using MVC.View;
using MVC.Controller;

namespace UseMVCApplication
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            UsersView view = new UsersView();

            IList users = new ArrayList();

            users.Add(new User("Vladimir", "Putin",
                 "122", "Gobernador de Russia",
                User.SexOfPerson.Male));
            users.Add(new User("Barack", "Obama",
                 "123", "Gobernador de USA",
                 User.SexOfPerson.Male));
            users.Add(new User("Stephen", "Harper",
                "124", "Gobernador de Canada",
                User.SexOfPerson.Male));
            users.Add(new User("Jean", "Charest",
                "125", "Gobernador de Quebec",
                 User.SexOfPerson.Male));
```

La principal ventaja de usar el patrón Modelo Vista Controlador (MVC) es que hace un enfoque muy estructurado sobre el proceso de diseño de la interfaz de usuario, que en sí mismo contribuye a escribir código limpio y comprobable, que será fácil de mantener y extender

El Modelo-Vista-Controlador es un patrón de diseño bien probado para resolver el problema de la separación de datos (modelo) e inquietudes de la interfaz de usuario (vista), de modo que los cambios en la interfaz de usuario no afecten el manejo de datos, y que los datos puedan ser cambiado sin impactar / cambiar la IU. El MVC resuelve este problema al desacoplar el acceso a datos y la capa de lógica empresarial de la interfaz de usuario y la interacción del usuario, al introducir un componente intermedio: el controlador. Esta arquitectura MVC permite la creación de componentes reutilizables dentro de un diseño de programa flexible (los componentes se pueden modificar fácilmente).