Spectacole

Enunț

Pentru o sală de spectacole, într-o anumită zi, există mai multe cereri de programare. Pentru fiecare cerere se cunosc următoarele: denumirea, ora de început, ora de sfârşit. Se cere să se construiască o planificare astfel încât numărul de conferințe programate să fie maxim şi să nu existe suprapuneri.

Rezolvare

Planificarea cerută se poate construi foarte simplu utilizând următorul algoritm:

1. Se sortează crescător spectacolele în funcție de ora de sfârşit.
2. Se selectează primul spectacol în lista de programări.
3. Se alege primul spectacol compatibil cu ultimul selectat (nu se suprapune) şi se adaugă în listă
4. Se continuă cu pasul 3 până la epuizarea listei.

Se poate demonstra foarte uşor că algoritmul prezentat conduce la o soluție optimă (număr maxim de spectacole).

Versiunea 0: Un mic prototip pentru testarea algoritmului

Pentru început vom construi o aplicație simplă pentru testarea algoritmului.

Aplicația va citi datele de la tastatură şi le va memora intr-o listă folosind clasa *System.Collections.ArrayList* şi o clasă ajutătoare pentru memorarea datelor despre un spectacol. După citirea datelor se aplică algoritmul prezentat anterior şi se afişează rezultatele pe ecran. Pentru sortarea spectacolelor am folosit o clasă ajutătoare care implementează interfața *IComparer* şi metoda *Sort* a clasei *ArrayList*.

Codul sursă pentru această variantă este:

```
using System;
using System.Collections;           // pentru ArrayList

namespace Spectacole.Test
{
      // clasa care contine informatiile despre
      // un spectacol (denumire, ora start, ora sfarsit)
      class Spectacol
      {
            // contructor pentru initializarea datelor
            public Spectacol(string denumire, int oraInceput, int oraSfarsit)
            {
                  Denumire = denumire;
                  OraInceput = oraInceput;
                  OraSfarsit = oraSfarsit;
            }

            // datele despre un spectacol (folosim campuri publice
            // pentru simplificare; in mod normal tebuiau incapsulate
            // in spatele unor proprietati)
            public string Denumire;
            public int OraInceput, OraSfarsit;
      }
```

```
class AppSpectacole
{
        // citirea listei de spectacole de la tastatura
        public static ArrayList CitireListaSpectacole()
        {
                // initializam lista
                ArrayList lista = new ArrayList();

                // citim numarul de spectacole
                int n;
                Console.Write("Numar spectacole:");
                n = int.Parse(Console.ReadLine());

                // citim spectacolele
                for (int i = 0; i < n; i++)
                {
                        string denumire;
                        int oraInceput, oraSfarsit;

                        // citim datele
                        Console.Write("Denumire:");
                        denumire = Console.ReadLine();

                        Console.Write("Ora inceput:");
                        oraInceput = int.Parse(Console.ReadLine());

                        Console.Write("Ora sfarsit:");
                        oraSfarsit = int.Parse(Console.ReadLine());

                        // adaugam spectacolul in lista
                        lista.Add(new Spectacol(denumire, oraInceput, oraSfarsit));
                }

                // returnam lista citita
                return lista;
        }

        // clasa utilizata pentru compararea spectacolelor in functie
        // de ora de sfarsit
        public class CompărareSpectacole : IComparer
        {
                int IComparer.Compare(object x, object y)
                {
                        if (((Spectacol)x).OraSfarsit < ((Spectacol)y).OraSfarsit)
                                return -1;      // x < y

                        if (((Spectacol)x).OraSfarsit == ((Spectacol)y).OraSfarsit)
                                return 0;       // x = y

                        return 1;               // x > y
                }
        }

        /// <summary>
        /// Functia pentru construirea programarii conform algoritmului
        /// </summary>
        /// <param name="lista">Lista de spectacole.</param>
        /// <returns>Lista cu spectacolele programate.</returns>
        /// <remarks>
        ///             Lista primita ca parametru va fi sortata de catre metoda
        ///             in functie de ora de sfarsit.
        ///             Rezultatul este o lista de referinte la obiectele din
        ///             lista initiala.
        /// </remarks>
        public static ArrayList Programare(ArrayList lista)
        {
                // Pasul 1: Sortarea listei in functie de ora de sfarsit
                lista.Sort(new CompărareSpectacole());

                // Pasul 2: Initializam lista de programate cu primul spectacol
                ArrayList rezultat = new ArrayList();
                rezultat.Add(lista[0]);

                // Pasul 3: Completarea listei de spectacole programate
                for (int i = 1; i < lista.Count; i++)
                        if (((Spectacol)rezultat[rezultat.Count - 1]).OraSfarsit <=
```

```
                                ((Spectacol)lista[i]).OraInceput) // daca nu se suprapun
                                rezultat.Add(lista[i]);

                        // returnam lista de spectacole programate
                        return rezultat;
                }

                // afiseaza pe ecran lista de spectacole
                public static void AfisareListaSpectacole(ArrayList lista)
                {
                        foreach(Spectacol spectacol in lista)
                                Console.WriteLine("{0} {1} - {2}",
                                        spectacol.Denumire,
                                        spectacol.OraInceput,
                                        spectacol.OraSfarsit);
                }

                static void Main(string[] args)
                {
                        // citim datele de intrare de la tastatura
                        ArrayList lista = CitireListaSpectacole();

                        // construim programarea optima
                        ArrayList programate = Programare(lista);

                        // afisam rezultatul
                        AfisareListaSpectacole(programate);
                }
        }
}
```

Deşi soluția prezentată rezolvă aparent problema, aceasta are foarte multe dezavantaje. În primul rând aplicația este foarte dificil de utilizat şi nu oferă nici un fel de validări. Datele trebuie introduse la fiecare rulare şi nu există posibilitatea de salvare a datelor sau de modificare a acestora după ce au fost introduse. Modelul foarte simplist utilizat nu oferă suficiente facilități pentru dezvoltări ulterioare.

Varianta completă este disponibilă aici.

Versiunea 1: Construirea unui model al problemei

În această variantă vom construi un model mai elaborat al problemei de rezolvat. Pentru aceasta vom adăuga două proiecte noi în cadrul soluției: un proiect de tip *Class Library* pentru model şi un nou proiect de tip consolă pentru testarea noului model.

Din analiza problemei rezultă şi din informațiile culese din implementarea prototipului putem deduce faptul că modelul problemei va conține două clase de bază: o clasă pentru memorarea datelor despre un spectacol şi o clasă pentru memorarea unei liste de spectacole şi obținerea rezultatului (care este tot o listă de spectacole).

Clasa *Spectacole* va rămâne aproape neschimbată. Vor fi adăugate doar proprietăți pentru încapsularea câmpurilor şi pentru efectuarea validărilor. O altă diferență notabilă este faptul că noua clasă va fi imutabilă, adică valorile vor putea fi stabilite numai la crearea obiectului. Aceasta va permite o simplificare a implementării clasei colecție.

Codul nou pentru clasa *Spectacol* este:

```
using System;

namespace Spectacole
{
```

```
    public class Spectacol
    {
            #region Constructor
            public Spectacol(string denumire, int oraInceput, int oraSfarsit)
            {
                    // validare date
                    if (denumire == null)
                            throw new ArgumentNullException("denumire",
                                    "Denumirea spectacolului este obligatorie.");

                    if (oraInceput < 0 || oraInceput > 23)
                            throw new ArgumentOutOfRangeException("oraInceput",
                                    "Ora de inceput trebuie sa fie intre 0 si 23");

                    if (oraSfarsit < 0 || oraSfarsit > 23)
                            throw new ArgumentOutOfRangeException("oraSfarsit",
                                    "Ora de sfarsit trebuie sa fie intre 0 si 23");

                    if (oraInceput >= oraSfarsit)
                            throw new ArgumentException(
                                    "Ora de sfarsit trebuie sa fie mai mare decat ora de
    inceput");

                    // atribuire valori
                    this.denumire = denumire;
                    this.oraInceput = oraInceput;
                    this.oraSfarsit = oraSfarsit;
            }
            #endregion

            #region Proprietati publice: Denumire, OraInceput, OraSfarsit
            public string Denumire
            {
                    get { return denumire; }
            }

            public int OraInceput
            {
                    get { return oraInceput; }
            }

            public int OraSfarsit
            {
                    get { return oraSfarsit; }
            }
            #endregion

            #region Metode publice: ECompatibilCu, ToString
            public bool ECompatibilCu(Spectacol spectacol)
            {
                    // daca celalalt spectacol nu exista atunci
                    // spunem prin conventie ca sunt compatibile
                    if (spectacol == null)
                            return true;

                    // sunt compatibile daca nu au portiuni comune
                    return
                            OraSfarsit <= spectacol.OraInceput ||
                            OraInceput >= spectacol.OraSfarsit;
            }

            public override string ToString()
            {
                    return string.Format("{0} [{1},{2}]",
                            Denumire, OraInceput, OraSfarsit);
            }
            #endregion

            #region Atribute private: denumire, oraInceput, oraSfarsit
            string denumire;
            int oraInceput, oraSfarsit;
            #endregion
    }
}
```

Pentru clasa care va memora lista de spectacole avem următoarele cerințe:

a) să poată fi utilizată ca o colecție normală de .Net (sa permită adăugări, modificări, ştergeri, acces prin index, enumerare cu *foreach*, etc.) şi să ofere suport direct pentru tipul *Spectacole* (să nu mai fie nevoie de cast-uri);
b) să permită obținerea soluției sub forma unei colecții similare;
c) să publice evenimente pentru operațiile importante;

Punctul a) va fi rezolvat construirea unei clase *ListaSpectacole* care va implementa interfața *IList*. Pentru a permite utilizarea directă a clasei *Spectacol* în locul clasei generice *Object* interfața va fi implementată explicit şi se vor adăuga metode similare dar care utilizează clasa specifică. Clasa va stoca intern datele intr-un ArrayList.

Pentru a permite obținerea rapidă a unei soluții, elementele listei vor fi ținute în permanență sortate după ora de sfârşit (prin intermediul operației de adăugare şi ştergere).

Colecția va publica evenimente pentru operațiile de adăugare, ştergere şi modificare spectacol.

Codul pentru clasa colecție este:

```
using System;
using System.Collections;

namespace Spectacole.Model
{
	#region Parametrii pentru delegati
	public class AdaugareEventArgs
	{
		public AdaugareEventArgs(Spectacol spectacol)
		{
			Spectacolul = spectacol;
		}
		public readonly Spectacol Spectacolul;
	}

	public class ModificareEventArgs
	{
		public ModificareEventArgs(Spectacol spectacolVechi, Spectacol spectacolNou)
		{
			SpectacolVechi = spectacolVechi;
			SpectacolNou   = spectacolNou;
		}
		public readonly Spectacol SpectacolVechi, SpectacolNou;
	}

	public class StergereEventArgs
	{
		public StergereEventArgs(Spectacol spectacol)
		{
			Spectacolul = spectacol;
		}
		public readonly Spectacol Spectacolul;
	}
	#endregion

	#region Delegatii folositi pentru evenimente
	public delegate void AdaugareEventHandler  (object sender, AdaugareEventArgs e);
	public delegate void ModificareEventHandler(object sender, ModificareEventArgs e);
	public delegate void StergereEventHandler  (object sender, StergereEventArgs e);
	#endregion

	public class ListaSpectacole : IList
	{
		#region Constructor
		public ListaSpectacole()
		{
```

```
        // initializam lista
        lista = new ArrayList();
}
#endregion

#region Interfata publica pentru accesare si modificare colectie
public void AdaugaSpectacol(Spectacol spectacol)
{
        ((IList)this).Add(spectacol);
}

public void StergeSpectacol(int index)
{
        ((IList)this).RemoveAt(index);
}

public Spectacol this[int index]
{
        get
        {
                return (Spectacol)((IList)this)[index];
        }
        set
        {
                ((IList)this)[index] = value;
        }
}

public int NumarSpectacole
{
        get
        {
                return lista.Count;
        }
}
#endregion

#region Metoda de planificare
public ListaSpectacole Planificare()
{
        // construim lista optima conform algoritmului
        ListaSpectacole rezultat = new ListaSpectacole();
        Spectacol ultimulSpectacol = null;

        foreach(Spectacol spectacol in this)
                if (spectacol.ECompatibilCu(ultimulSpectacol))
                {
                        rezultat.AdaugaSpectacol(spectacol);
                        ultimulSpectacol = spectacol;
                }

        return rezultat;
}
#endregion

#region Evenimente: Declaratii si metode pentru publicare
public event AdaugareEventHandler Adaugare;
protected virtual void OnAdaugare(Spectacol spectacol)
{
        if (Adaugare != null)
                Adaugare(this, new AdaugareEventArgs(spectacol));
}

public event ModificareEventHandler Modificare;
protected virtual void OnModificare(
        Spectacol spectacolVechi, Spectacol spectacolNou)
{
        if (Modificare != null)
                Modificare(this, new ModificareEventArgs(
                        spectacolNou, spectacolVechi));
}

public event StergereEventHandler Stergere;
protected virtual void OnStergere(Spectacol spectacol)
{
        if (Stergere != null)
                Stergere(this, new StergereEventArgs(spectacol));
```

```
        }
        #endregion

        #region Implementare explicita IList
        bool IList.IsReadOnly
        {
                get { return false; }
        }

        object IList.this[int index]
        {
                get
                {
                        return lista[index];
                }
                set
                {
                        // verificam ca obiectul primit este un spectacol
                        if (!(value is Spectacol))
                                throw new ArgumentException(
                                        "Colectia nu suporta decat obiecte de tip
Spectacol.",
                                        "value");

                        OnModificare((Spectacol)lista[index], (Spectacol)value);

                        /// HACK: In mod normal nu ar trebui sa facem asta pentru nu
respecta semantica operatiei.

                        // folosim functiile definite anterior
                        // pentru pastrarea proprietatii listei
                        ((IList)this).RemoveAt(index);
                        ((IList)this).Add(value);
                }

        }

        void IList.RemoveAt(int index)
        {
                Spectacol spectacol = (Spectacol)lista[index];
                lista.RemoveAt(index);
                OnStergere(spectacol);
        }

        void IList.Insert(int index, object value)
        {
                throw new NotSupportedException(
                        "Colectia nu suporta inserarea pe o anumita pozitie.");
        }

        void IList.Remove(object value)
        {
                // verificam ca obiectul primit este un spectacol
                if (!(value is Spectacol))
                        throw new ArgumentException(
                                "Colectia nu suporta decat obiecte de tip Spectacol.",
                                "value");

                // daca elementul exista
                if (lista.IndexOf(value) >= 0)
                {
                        // tinem minte elementul care va fi sters
                        Spectacol spectacol = (Spectacol)lista[lista.IndexOf(value)];

                        // stergem obiectul din lista
                        lista.Remove(value);

                        // si publicam evenimentul
                        OnStergere(spectacol);
                }
        }

        bool IList.Contains(object value)
        {
                // verificam ca obiectul primit este un spectacol
                if (!(value is Spectacol))
                        throw new ArgumentException(
```

```
                        "Colectia nu suporta decat obiecte de tip Spectacol.",
                        "value");

        return lista.Contains(value);
}

void IList.Clear()
{
        // copiem referintele
        ArrayList listaDeSters = (ArrayList)lista.Clone();

        // stergem referintele
        lista.Clear();

        // publicam evenimentele pentru toate cele sterse
        foreach(Spectacol spectacol in listaDeSters)
                OnStergere(spectacol);

        // si curatam ce a mai ramas
        listaDeSters.Clear();
}

int IList.IndexOf(object value)
{
        // verificam ca obiectul primit este un spectacol
        if (!(value is Spectacol))
                throw new ArgumentException(
                        "Colectia nu suporta decat obiecte de tip Spectacol.",
                        "value");

        return lista.IndexOf(value);
}

int IList.Add(object value)
{
        // verificam ca spectacolul este diferit de null
        if (value == null)
                throw new ArgumentNullException("value",
                        "Spectacolul nu poate fi null.");

        // verificam ca obiectul primit este un spectacol
        if (!(value is Spectacol))
                throw new ArgumentException(
                        "Colectia nu suporta decat obiecte de tip Spectacol.",
                        "value");

        // convertim obiectul primit
        Spectacol spectacol = (Spectacol)value;

        // inseram spectacolul in pozitia corecta in lista
        // (a.i. lista sa ramana ordonata in functie de
        // ora de sfarsit)
        for (int i = 0; i < lista.Count; i++)
                if (spectacol.OraSfarsit < ((Spectacol)lista[i]).OraSfarsit)
                {
                        lista.Insert(i, spectacol);
                        OnAdaugare(spectacol);
                        return i;
                }

        // daca am ajuns aici inseamna ca lista e vida sau
        // ca toate spectacolele au ora de sfarsit mai mica
        // decat ora de sfarsit a specacolului de adaugat
        // deci spectacolul trebuie adaugat la sfarsit
        int index = lista.Add(spectacol);
        OnAdaugare(spectacol);
        return index;
}

bool IList.IsFixedSize
{
        get { return false; }
}
#endregion

#region Implementare explicita ICollection
bool ICollection.IsSynchronized
```

```
		{
			get { return lista.IsSynchronized; }
		}

		int ICollection.Count
		{
			get { return lista.Count; }
		}

		void ICollection.CopyTo(Array array, int index)
		{
			lista.CopyTo(array, index);
		}

		object ICollection.SyncRoot
		{
			get { return lista.SyncRoot; }
		}
		#endregion

		#region Implementare explicita IEnumerable
		IEnumerator IEnumerable.GetEnumerator()
		{
			return lista.GetEnumerator();
		}
		#endregion

		#region Atribute private: lista
		// lista folosita pentru stocarea datelor
		public ArrayList lista;
		#endregion
	}
}
```

Pentru a oferi o modalitate cât mai flexibilă de salvare/restaurare, vom defini un contract numit *IPersistentaSpectacole* care defineşte operaţiile necesare pentru salvare/restaurare. Iniţial vom construi doar o clasă care să permită salvarea/restaurarea în/din fişiere text. Sistemul poate fi extins prin crearea in aplicaţiile client de noi clase care implementează interfaţa *IPersistentaSpectacole*.

Pentru testarea noului modelului am construit o nouă aplicaţie de consolă:

```
using System;

using Spectacole.Model;

namespace Spectacole
{
	class AppSpectacole
	{
		[STAThread]
		static void Main(string[] args)
		{
			// obtinere parametri din linia de comanda
			// (in VS se seteaza de la Project->Spectacole.Console Properties... si
			// selectati Configuration Properties -> Debugging -> Command line args)
			if (args.Length != 1)
			{
				Console.WriteLine("Sintaxa este: spectacole <nume_fisier>");
				return;
			}
			string numeFisier = args[0];

			// citire date din fisier
			ListaSpectacole lista = new PersistentaFisier(numeFisier).Restaurare();

			// afisare spectacole
			Console.WriteLine("--- Lista spectacole ---");
			foreach(Spectacol spectacol in lista)
				Console.WriteLine(spectacol);
```

```
                // afisare programare optima
                Console.WriteLine("\n--- Programarea optima ---");
                ListaSpectacole rezultat = lista.Planificare();
                foreach(Spectacol spectacol in rezultat)
                        Console.WriteLine(spectacol);

                // salvare planificare optima in fisier
                new PersistentaFisier(numeFisier + "rez.txt").Salvare(rezultat);
        }
}
}
```

În mod normal aici ar fi trebuit testate toate operațiile oferite de către model, dar vom sări peste partea asta deoarece le vom testa în versiunea următoare a aplicației.

Versiunea completă este disponibilă aici.

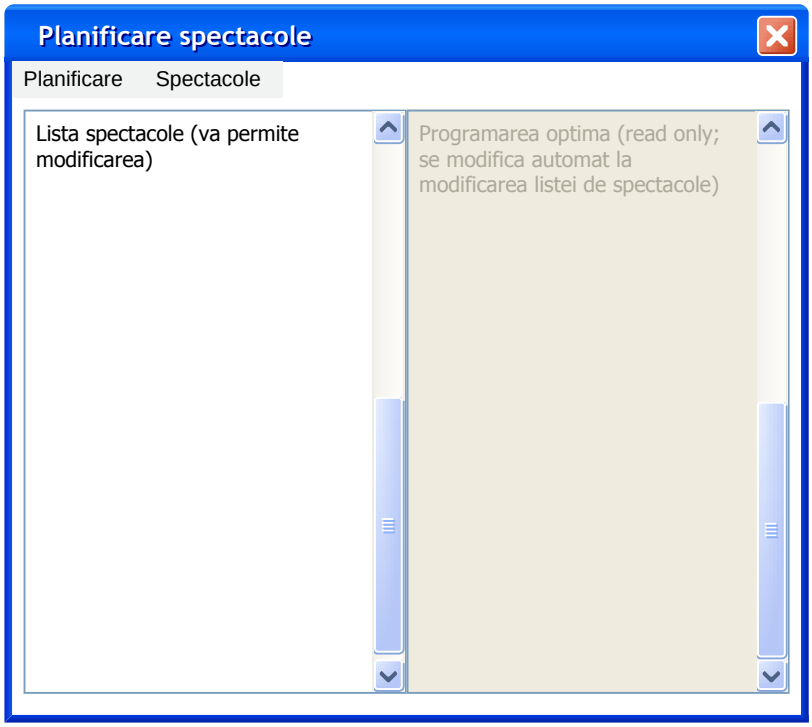
Versiunea 2: Aplicație Windows

În această versiune vom reutiliza modelul pentru a construi o aplicație Windows care să permită o utilizare mai facilă a facilităților oferite de model.

Pentru început vom construi o aplicație simplă care să suporte următoarele operații:

- creare listă nouă, salvare, restaurare
- adăugare, ştergere şi modificare spectacole
- vizualizare şi salvare planificare optimă

Interfața va fi constituită în principal dintr-o fereastră cu două controale de tip *ListBox* ca în figura următoare:



Pentru construirea aplicației vom parcurge următorii paşi:

1. Modificăm forma creată de VS astfel:
- mutăm metoda Main intr-o clasă nouă

- adăugam un câmp de tip *ListaSpectacole* în formă şi modificăm constructorul a.î. să primească ca parametru o referință la un obiect de tip *ListaSpectacole*
- în Main construim modelul şi trimitem o referință ca parametru în constructor

2. Adăugăm elementele folosite în interfață
   a. meniul principal si submeniurile
   b. cele două liste (de tip *ListBox*)
   c. bara de separare (*Splitter*)
3. Adăugăm o funcție pentru afişarea datelor din model:

```
private void AfisareDate()
{
        // readaugam elementele in lista de spectacole
        lbSpectacole.Items.Clear();
        foreach(Spectacol spectacol in Model)
                lbSpectacole.Items.Add(spectacol);

        // readaugam elementele in lista de spectacole planificate
        ListaSpectacole planificate = Model.Planificare();
        lbPlanificare.Items.Clear();
        foreach(Spectacol spectacol in planificate)
                lbPlanificare.Items.Add(spectacol);
}
```

4. Adăugăm două proprietăți pentru a ține minte numele fişierului curent şi dacă a fost modificat:

```
#region Proprietati forma: Modificat, NumeFisier
private string numeFisier;
private string NumeFisier
{
        get { return numeFisier;  }
        set { numeFisier = value; }
}

private bool modificat;
private bool Modificat
{
        get { return modificat;  }
        set { modificat = value; }
}
#endregion
```

5. Adăugăm handlerele pentru operații cu fişiere:

```
private void miPlanificareNoua_Click(object sender, System.EventArgs e)
{
    Model.StergeLista();   // stergem spectacolele
    Modificat = false;            // resetam indicatorii
    NumeFisier = string.Empty;
    AfisareDate();                // si afisam din nou datele
}

private void miPlanificareDeschideFisier_Click(object sender, System.EventArgs e)
{
    // atentionam utilizatorul ca va pierde modificarile daca continua
    if (Modificat)
    {
            DialogResult rezultat = MessageBox.Show(this,
                    "Doriti sa incarcati un fisier nou si sa pierdeti modificarile?",
                    "Planificare             Spectacole",             MessageBoxButtons.YesNo,
    MessageBoxIcon.Warning);

            // utilizatorul refuza atunci anulam operatia
            if (rezultat == DialogResult.No)
                    return;
    }

    Model.StergeLista();   // stergem spectacolele
```

```
    // obtinem numele fisierului
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "Fisiere text (*.txt)|*.txt|Fisiere oarecare(*.*)|*.*";
    if (ofd.ShowDialog(this) == DialogResult.OK)
    {
            // daca userul nu a anulat comanda
            // citim datele din fisier
            Model = new PersistentaFisier(ofd.FileName).Restaurare();

            Modificat = false;            // resetam indicatorii
            NumeFisier = ofd.FileName;

            AfisareDate();                // si afisam din nou datele
    }
}

private void miPlanificareSalvare_Click(object sender, System.EventArgs e)
{
    if (Modificat) // doar daca exista modificari
    {
            if (NumeFisier != string.Empty)       // daca avem un nume de fisier
                    // atunci il salvam direct
                    new PersistentaFisier(NumeFisier).Salvare(Model);
            else
                    // altfel apelam "Salvare Ca..."
                    miPlanificareSalvareCa_Click(null, new EventArgs());

            Modificat = false;     // si resetam indicatorul de modificat
    }
}

private void miPlanificareSalvareCa_Click(object sender, System.EventArgs e)
{
    // construim dialogul pentru alegerea fisierului
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Fisiere text (*.txt)|*.txt|Fisiere oarecare(*.*)|*.*";

    // daca userul nu a anulat comanda
    if (sfd.ShowDialog(this) == DialogResult.OK)
            // salvam datele in fisier
            new PersistentaFisier(sfd.FileName).Salvare(Model);
}
```

6. Construim o nouă formă de tip dialog numită *EditareSpectacol* pentru adăugare/modificare spectacol cu următoarele controale:
    a. Trei TextBox-uri şi trei Label-uri pentru introducerea datelor
    b. Două butoane pentru acceptare/anularea modificărilor.
    c. Un *ErrorProvider* pentru validare
7. Adăugăm o proprietate de tip *Spectacol*:

```
    Spectacol spectacolCurent;
    public Spectacol SpectacolCurent
    {
            get { return spectacolCurent; }
            set
            {
                    // verificam ca avem un spectacol valid
                    if (value == null)
                            throw new ArgumentNullException("value",
                                    "Proprietatea nu poate fi asignata cu valoarea null");

                    spectacolCurent = value;
            }
    }
```

8. Adăugăm constructorii pentru formă:

```
    public FormaEditare()
    {
            // intitializarea componentelor adaugate
```

```
            // cu designer-ul din VS
            InitializeComponent();

            // daca nu avem nici un parametru
            // atunci inseamna ca trebuie sa creem
            // un spectacol nou
            Text = "Adaugare Spectacol";
    }

    public FormaEditare(Spectacol spectacol)
    {
            // intitializarea componentelor adaugate
            // cu designer-ul din VS
            InitializeComponent();

            // daca avem un parametru atunci initializam
            // forma pentru modificare

            Text = "Modificare Spectacol";

            txtDenumire.Text   = spectacol.Denumire;
            txtOraInceput.Text = spectacol.OraInceput.ToString();
            txtOraSfarsit.Text = spectacol.OraSfarsit.ToString();

            SpectacolCurent = spectacol;
        }
```

9. Adăugăm codul pentru validare:

```
private void txtDenumire_Validating(object sender, System.ComponentModel.CancelEventArgs e)
{
    if (txtDenumire.Text == null || txtDenumire.Text == string.Empty)
            errorProvider.SetError(txtDenumire, "Un spectacol trebuie sa aiba o denumire.");
    else
            errorProvider.SetError(txtDenumire, "");
}

private void txtOraInceput_Validating(object sender, System.ComponentModel.CancelEventArgs e)
{
    try
    {
            int oraInceput = int.Parse(txtOraInceput.Text);

            if (oraInceput < 0 || oraInceput > 23)
                    throw new Exception("Ora de inceput trebuie sa fie intre 0 si 23.");
    }
    catch(FormatException)
    {
            errorProvider.SetError(txtOraInceput, "Ora de inceput trebuie sa fie un intreg.");
    }
    catch(Exception ex)
    {
            errorProvider.SetError(txtOraInceput, ex.Message);
    }
    errorProvider.SetError(txtOraInceput, "");
}

private void txtOraSfarsit_Validating(object sender, System.ComponentModel.CancelEventArgs e)
{
    try
    {
            int oraSfarsit = int.Parse(txtOraSfarsit.Text);

            if (oraSfarsit < 0 || oraSfarsit > 23)
                    throw new Exception("Ora de sfarsit trebuie sa fie intre 0 si 23.");
    }
    catch(FormatException)
    {
            errorProvider.SetError(txtOraSfarsit, "Ora de sfarsit trebuie sa fie un intreg.");
    }
    catch(Exception ex)
    {
            errorProvider.SetError(txtOraSfarsit, ex.Message);
    }
    errorProvider.SetError(txtOraSfarsit, "");
```

```
}
```

10. Adăugăm codul pentru butonul de acceptare:

```
private void btnOK_Click(object sender, System.EventArgs e)
{
    // validam datele
    if (txtDenumire.Text == null || txtDenumire.Text == string.Empty)
    {
            errorProvider.SetError(txtDenumire, "Un spectacol trebuie sa aiba o denumire.");
            return;
    }

    int oraInceput = 0;
    int oraSfarsit = 0;

    try
    {
            oraInceput = int.Parse(txtOraInceput.Text);

            if (oraInceput < 0 || oraInceput > 23)
                    throw new Exception("Ora de inceput trebuie sa fie intre 0 si 23.");
    }
    catch(FormatException)
    {
            errorProvider.SetError(txtOraInceput, "Ora de inceput trebuie sa fie un intreg.");
            return;
    }
    catch(Exception ex)
    {
            errorProvider.SetError(txtOraInceput, ex.Message);
            return;
    }

    try
    {
            oraSfarsit = int.Parse(txtOraSfarsit.Text);

            if (oraSfarsit < 0 || oraSfarsit > 23)
                    throw new Exception("Ora de sfarsit trebuie sa fie intre 0 si 23.");
    }
    catch(FormatException)
    {
            errorProvider.SetError(txtOraSfarsit, "Ora de sfarsit trebuie sa fie un intreg.");
            return;
    }
    catch(Exception ex)
    {
            errorProvider.SetError(txtOraSfarsit, ex.Message);
            return;
    }

    if (oraInceput >= oraSfarsit)
    {
            errorProvider.SetError(txtOraSfarsit,
                    "Ora de sfarsit trebuie sa fie mai mare decat ora de inceput.");

            return;
    }

    // daca am ajuns aici inseamna ca datele sunt corecte
    SpectacolCurent = new Spectacol(
            txtDenumire.Text, oraInceput, oraSfarsit);
    }
```

11. Revenim la forma principală şi adăugăm handlerele pentru butoanele de modificare:

```
private void miSpectacoleAdauga_Click(object sender, System.EventArgs e)
{
    // construim dialogul de editare
    FormaEditare forma = new FormaEditare();
```

```
    // afisam dialogul si actionam daca userul apasa ok
    if (forma.ShowDialog(this) == DialogResult.OK)
    {
            // adaugam spectacolul in model
            Model.AdaugaSpectacol(forma.SpectacolCurent);

            // schimbam indicatorii
            Modificat = true;

            // redesenam ecranul
            AfisareDate();
    }
}

private void miSpectacoleModifica_Click(object sender, System.EventArgs e)
{
    // daca avem un spectacol selectat
    if (lbSpectacole.SelectedIndex >= 0)
    {
            // construim dialogul de editare
            FormaEditare forma = new FormaEditare((Spectacol)lbSpectacole.SelectedItem);

            // afisam dialogul si actionam daca userul apasa ok
            if (forma.ShowDialog(this) == DialogResult.OK)
            {
                    // aflam indexul spectacolului de modificat
                    int index = ((IList)Model).IndexOf(lbSpectacole.SelectedItem);

                    // si il modificam
                    Model[index] = forma.SpectacolCurent;

                    // schimbam indicatorii
                    Modificat = true;

                    // redesenam ecranul
                    AfisareDate();
            }
    }
}

private void miSpectacoleSterge_Click(object sender, System.EventArgs e)
{
    // daca avem un spectacol selectat
    if (lbSpectacole.SelectedIndex >= 0)
    {
            if (MessageBox.Show(this, "Doriti sa stergeti spectacolul?","Stergere Spectacol",
                    MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                    // aflam indexul spectacolului de sters
                    int index = ((IList)Model).IndexOf(lbSpectacole.SelectedItem);

                    // si il stergem
                    Model.StergeSpectacol(index);

                    // schimbam indicatorii
                    Modificat = true;

                    // redesenam ecranul
                    AfisareDate();
            }
    }
}
```

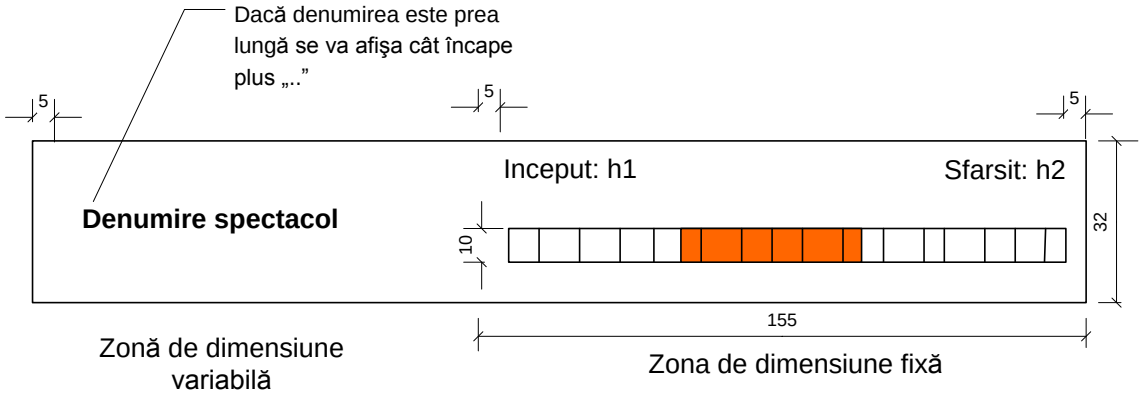
În acest moment avem o interfață funcțională pentru aplicația de planificat spectacole.

Versiunea completă este disponibilă aici.

Versiunea 3: Îmbunătățirea interfeței grafice

Desenarea manuală a elementelor listei

În primul rând vom reface aspectul listelor prin desenarea manuală a elementelor. Forma propusă pentru elementele listelor este:



Pentru a desena manual elementele trebuie setate proprietățile DrawMode = OwnerDrawFixed şi ItemHeight = 32 (în cazul în care se doreşte desenarea de elemente cu înălțime variabilă se ca seta proprietatea DrawMode pe OwnerDrawVariable). Desenarea elementelor se realizează prin tratarea evenimentului DrawItem:

```
private void DesenareElementLista(object sender, System.Windows.Forms.DrawItemEventArgs e)
{
    const int DimZonaFixa = 155;

    if (e.Index < 0)       // in mod normal nu ar trebui sa se intample :)
            return;

    // obtinem o referinta la spectacolul curent
    Spectacol s = (Spectacol)((ListBox)sender).Items[e.Index];

    // salvam referinta la suprafata pe care vom desena
    Graphics g = e.Graphics;

    // construim pensulele
    Brush bFundal = new SolidBrush(e.BackColor);
    Brush bText   = new SolidBrush(e.ForeColor);

    // desenam fundalul
    e.DrawBackground();
    e.DrawFocusRectangle();

    // desenam linia de sfarsit
    g.DrawLine(new  Pen(bText,  1),  e.Bounds.Left,  e.Bounds.Bottom  -  1,  e.Bounds.Right,
    e.Bounds.Bottom - 1);


    // ----- Desenare denumire spectacol ----- //

    // construim fontul cu care vom desena denumirea
    Font fDenumire = new Font(e.Font, FontStyle.Bold);

    // zona in care desenam
    Rectangle rectDenumire = new Rectangle(
            e.Bounds.Left, e.Bounds.Top,
            e.Bounds.Width - DimZonaFixa - 3,
            e.Bounds.Height);

    // formatarea pentru denumire
    StringFormat formatDenumire =  new StringFormat();
```

```
    formatDenumire.Alignment     = StringAlignment.Near;
    formatDenumire.LineAlignment = StringAlignment.Center;
    formatDenumire.FormatFlags = StringFormatFlags.NoWrap;
    formatDenumire.Trimming = StringTrimming.EllipsisCharacter;

    // desenam denumirea
    g.DrawString(s.Denumire, fDenumire, bText, rectDenumire, formatDenumire);

    // ----- Desenare text ore ----- //

    // zona in care desenam
    RectangleF rectTextOre = new RectangleF(
            e.Bounds.Right - DimZonaFixa,
            e.Bounds.Top + 2,
            DimZonaFixa - 6, 12);

    // formatarea pentru sfarsit (la dreapta)
    StringFormat formatSfarsit =  new StringFormat();
    formatSfarsit.Alignment     = StringAlignment.Far;

    // desenam orele
    g.DrawString("Inceput: " + s.OraInceput, e.Font, bText, rectTextOre);
    g.DrawString("Sfarsit: " + s.OraSfarsit, e.Font, bText, rectTextOre, formatSfarsit);

    // ----- Desenare grafica ore ----- //

    int start = e.Bounds.Right - DimZonaFixa;

    // desenam fundalul alb
    g.FillRectangle(Brushes.White, start, e.Bounds.Bottom - 15, 6*24, 10);

    // desenam zona rosie
    g.FillRectangle(Brushes.Red,
            start+ s.OraInceput * 6,
            e.Bounds.Bottom - 15,
            (s.OraSfarsit - s.OraInceput) * 6, 10);

    // desenam gradatiile
    for (int i = 0; i < 24; i++)
            g.DrawLine(Pens.Black,
                    start + i * 6,
                    e.Bounds.Bottom - 15,
                    start + i * 6,
                    e.Bounds.Bottom - 5);

    // desenam conturul
    g.DrawRectangle(Pens.Black, start, e.Bounds.Bottom - 15, 145, 10);
}
```

Afişarea indicatorilor

Pentru planificarea optimă vom afişa principalii indicatori sub formă de text şi sub formă de grafic. Pentru aceasta adăugăm un control de tip *Panel* şi tratăm evenimentul *Paint* astfel:

```
private void pnlDetaliiProgramare_Paint(object sender, System.Windows.Forms.PaintEventArgs e)
{
    // ----- Calcul indicatori ----- //

    // obtinem programarea optima
    ListaSpectacole programate = Model.Planificare();

    // calculam numarul de ore ocupate
    int nrOreOcupate = 0;
    foreach(Spectacol s in programate)
            nrOreOcupate += s.OraSfarsit - s.OraInceput;

    // calculam numarul de spectacole
    int nrSpectacole = Model.NumarSpectacole;
    int nrProgramate = programate.NumarSpectacole;

    // calcul procent
    int procSpectacole = 0;
    if (nrSpectacole != 0)
```

```
            procSpectacole = (int)((nrProgramate*100) / nrSpectacole);

        // ----- Afisare text indicatori ----- //

        // obtinem o reerinta la suprafata controlului
        Graphics g = e.Graphics;

        Brush bText = new SolidBrush(pnlDetaliiProgramare.ForeColor);

        // formatarea pentru texte
        StringFormat formatText  =  new StringFormat();
        formatText.Alignment     = StringAlignment.Near;
        formatText.LineAlignment = StringAlignment.Center;
        formatText.FormatFlags   = StringFormatFlags.NoWrap;
        formatText.Trimming      = StringTrimming.EllipsisCharacter;

        // zonele in care afisam textele
        int latime = (int)((pnlDetaliiProgramare.Width - 10)/2);

        Rectangle rectNrTotal = new Rectangle(5, 5, pnlDetaliiProgramare.Width - 10, 15);
        Rectangle rectNrProgr = new Rectangle(5, 25, pnlDetaliiProgramare.Width - 10, 15);

        Rectangle rectProcOre = new Rectangle(5, 45, latime, 15);
        Rectangle rectProcNr  = new Rectangle(latime, 45, latime, 15);

        // desenam textele
        g.DrawString("Numar total de spectacole: " + nrSpectacole,
                pnlDetaliiProgramare.Font, bText,
                rectNrTotal, formatText);

        g.DrawString("Numar de spectacole programate: " + nrProgramate,
                pnlDetaliiProgramare.Font, bText,
                rectNrProgr, formatText);

        g.DrawString("% ore ocupate: " + (int)((nrOreOcupate*100) / 24),
                pnlDetaliiProgramare.Font, bText,
                rectProcOre, formatText);

        g.DrawString("% spectacole programate: " + procSpectacole,
                pnlDetaliiProgramare.Font, bText,
                rectProcNr, formatText);

        // ----- Afisare grafice ----- //

        g.FillPie(Brushes.Red, 5, 65, 60, 60, 0, 15*nrOreOcupate);
        g.DrawPie(Pens.Black, 5, 65, 60, 60, 0, 15*nrOreOcupate);
        g.FillPie(Brushes.White, 5, 65, 60, 60, 0, -15*(24-nrOreOcupate));
        g.DrawPie(Pens.Black, 5, 65, 60, 60, 0, -15*(24-nrOreOcupate));

        int unghi = 0;
        if (nrSpectacole != 0)
                unghi = (int)((360 * nrProgramate) / nrSpectacole);

        g.FillPie(Brushes.Red, 10 + latime, 65, 60, 60, 0, unghi);
        g.DrawPie(Pens.Black, 10 + latime, 65, 60, 60, 0, unghi);
        g.FillPie(Brushes.White, 10 + latime, 65, 60, 60, 0, unghi - 360);
        g.DrawPie(Pens.Black, 10 + latime, 65, 60, 60, 0, unghi - 360);
}
```

Adăugare bară de stare

Pentru afişarea indicatorilor (fişierul curent, modificat) vom adăuga o bară de stare (*StatusBar*) cu proprietățile următoare:

- Name: statusBar, Text gol, ShowPanels: true
- Panels-> 2 paneluri: sbpNumeFisier (AutoSize: Spring) şi sbpModificat

Pentru a asocia o imagine unui element din toolbar procedăm în felul următor:

- adăugăm un control de tip listă de imagini (*ImageList*)
- adăugăm imaginea dorită în listă
- modificăm proprietatea *Modificat* pentru a seta automat imaginea astfel:

```
private bool Modificat
{
        get { return modificat;  }
        set
        {
                modificat = value;

                if (modificat)
                {
                        sbpModificat.Text = "Modificat";
                        sbpModificat.Icon               =               Icon.FromHandle(new
                Bitmap(ilstStatusBar.Images[0]).GetHicon());
                }
                else
                {
                        sbpModificat.Text = "";
                        sbpModificat.Icon = null;
                }
        }
}
```

Restructurare cod

Pentru a permite implementarea facilă în continuare, vom restructura un pic codul existent:

- vom utiliza evenimentele modelului pentru redesenare
- vom grupa operațiile activate de meniu în funcții separate

Utilizarea evenimentelor modelului se va face în următorii paşi:

a) se adaugă în constructor handlerele pentru evenimente:

```
// legare evenimente model
Model.Adaugare +=new AdaugareEventHandler(Model_Adaugare);
Model.Modificare +=new ModificareEventHandler(Model_Modificare);
Model.Stergere += new StergereEventHandler(Model_Stergere);
```

b) se adaugă operațiile necesare în handlere

```
private void Model_Adaugare(object sender, AdaugareEventArgs e)
{
        Modificat = true;
        AfisareDate();
}

private void Model_Modificare(object sender, ModificareEventArgs e)
{
        Modificat = true;
        AfisareDate();
}

private void Model_Stergere(object sender, StergereEventArgs e)
{
        Modificat = true;
        AfisareDate();
}
```

c) se elimină apelurile necesare din celelalte handlere

Grupăm operațiile în metode separate astfel:

```
#region Operatii fisiere
private void FisierNou()
{
    Model.StergeLista();   // stergem spectacolele
    Modificat = false;            // resetam indicatorii
    NumeFisier = string.Empty;
}

private void DeschideFisier()
{
```

```
    // atentionam utilizatorul ca va pierde modificarile daca continua
    if (Modificat)
    {
            DialogResult rezultat = MessageBox.Show(this,
                    "Doriti sa incarcati un fisier nou si sa pierdeti modificarile?",
                    "Planificare             Spectacole",             MessageBoxButtons.YesNo,
    MessageBoxIcon.Warning);

            // utilizatorul refuza atunci anulam operatia
            if (rezultat == DialogResult.No)
                    return;
    }

    Model.StergeLista();   // stergem spectacolele

    // obtinem numele fisierului
    OpenFileDialog ofd = new OpenFileDialog();
    ofd.Filter = "Fisiere text (*.txt)|*.txt|Fisiere oarecare(*.*)|*.*";
    if (ofd.ShowDialog(this) == DialogResult.OK)
    {
            // daca userul nu a anulat comanda
            // citim datele din fisier
            Model = new PersistentaFisier(ofd.FileName).Restaurare();

            Modificat = false;            // resetam indicatorii
            NumeFisier = ofd.FileName;
    }
}

private void SalvareFisier()
{
    if (Modificat) // doar daca exista modificari
    {
            if (NumeFisier != string.Empty)       // daca avem un nume de fisier
                    // atunci il salvam direct
                    new PersistentaFisier(NumeFisier).Salvare(Model);
            else
                    // altfel apelam "Salvare Ca..."
                    SalvareFisierCa();

            Modificat = false;     // si resetam indicatorul de modificat
    }
}

private void SalvareFisierCa()
{
    // construim dialogul pentru alegerea fisierului
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Fisiere text (*.txt)|*.txt|Fisiere oarecare(*.*)|*.*";

    // daca userul nu a anulat comanda
    if (sfd.ShowDialog(this) == DialogResult.OK)
            // salvam datele in fisier
            new PersistentaFisier(sfd.FileName).Salvare(Model);
}
#endregion

#region Operatii spectacole
void AdaugaSpectacol()
{
    // construim dialogul de editare
    FormaEditare forma = new FormaEditare();

    // afisam dialogul si actionam daca userul apasa ok
    if (forma.ShowDialog(this) == DialogResult.OK)
    {
            // adaugam spectacolul in model
            Model.AdaugaSpectacol(forma.SpectacolCurent);
    }
}

void ModificaSpectacol()
{
    // daca avem un spectacol selectat
    if (lbSpectacole.SelectedIndex >= 0)
    {
            // construim dialogul de editare
```

```
            FormaEditare forma = new FormaEditare((Spectacol)lbSpectacole.SelectedItem);

            // afisam dialogul si actionam daca userul apasa ok
            if (forma.ShowDialog(this) == DialogResult.OK)
            {
                // aflam indexul spectacolului de modificat
                int index = ((IList)Model).IndexOf(lbSpectacole.SelectedItem);

                // si il modificam
                Model[index] = forma.SpectacolCurent;
            }
        }
    }

    void StergeSpectacol()
    {
        // daca avem un spectacol selectat
        if (lbSpectacole.SelectedIndex >= 0)
        {
            Spectacol spectacol = (Spectacol)lbSpectacole.SelectedItem;
            string mesaj = string.Format("Doriti sa stergeti spectacolul '{0}'?",
                spectacol.Denumire);

            if (MessageBox.Show(this, mesaj,"Stergere Spectacol",
                MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                // aflam indexul spectacolului de sters
                int index = ((IList)Model).IndexOf(spectacol);

                // si il stergem
                Model.StergeSpectacol(index);
            }
        }
    }
    #endregion
```

Adăugare bară de instrumente

Pentru a adăuga bara de instrumente parcurgem următorii paşi:

a) adăugăm o listă de imagini şi completăm lista cu imaginile corespunzătoare butoanelor;
b) adăugăm o bară de instrumente (*ToolBar*) şi completăm lista de butoane cu proprietățile aferente;
c) adăugăm codul pentru tratarea evenimentului (care refoloseşte funcțiile definite pentru operațiile din meniu):

```
    private void toolBar_ButtonClick(object sender,
    System.Windows.Forms.ToolBarButtonClickEventArgs e)
    {
        string tag = e.Button.Tag as string;
        switch (tag)
        {
            case "ListaNoua": FisierNou(); break;
            case "DeschidereFisier": DeschideFisier(); break;
            case "SalvareFisier": SalvareFisier(); break;
            case "Adauga": AdaugaSpectacol(); break;
            case "Modifica": ModificaSpectacol(); break;
            case "Sterge": StergeSpectacol(); break;
        }
    }
```

Adăugare meniu contextual

Pentru a adăuga un meniu contextual pentru lista de spectacole putem parcurge următorii paşi:

a) adăugăm controlul *ContextMenu* pe formular
b) adăugăm elementele corespunzătoare operațiilor

c) setăm proprietatea ContextMenu a listei a.î. să indice meniul creat
d) tratăm evenimentele generate de meniu:

```
private void meniuLista_Popup(object sender, System.EventArgs e)
{
        cmModifica.Enabled = (lbSpectacole.SelectedIndex >= 0);
        cmSterge.Enabled = (lbSpectacole.SelectedIndex >= 0);
}

private void cmAdauga_Click(object sender, System.EventArgs e)
{
        AdaugaSpectacol();
}

private void cmModifica_Click(object sender, System.EventArgs e)
{
        ModificaSpectacol();
}

private void cmSterge_Click(object sender, System.EventArgs e)
{
        StergeSpectacol();
}
```

Adăugare suport pentru tipărire

Pentru detalii despre tipărirea in Windows Forms vezi carte cap 9, articol MSDN şi Petzold cap 21.

În cazul aplicației noaste s-a construit o clasă *TiparSpectacole* derivată din *PrintDocument* care conține codul necesar desenării paginilor şi a fost adăugat codul pentru inițializarea tipăririi în aplicația principală.

Versiunea completă este disponibilă aici.

Versiunea 4: Suport pentru clipboard şi drag&drop

În această versiune vom face aplicația noastră interoperabilă cu alte aplicații Windows. Obiectivele sunt:

- posibilitatea de a copia un element in clipboard sub formă de text şi sub formă grafică
- posibilitatea de a insera unul sau mai multe spectacole din clipboard (în clipboard trebuie să fie în format text)
- posibilitatea de a face drag&drop cu spectacole între două instanțe ale aplicației noastre sau între o instanță a aplicației noastre şi Excel sau un program de editare grafică

Primul pas este adăugarea elementelor corespunzătoare în meniul aplicației şi în toolbar, precum şi crearea metodelor care vor implementa operațiile. Datorită faptului că elementele meniului nu suportă decât un singur shortcut, vom trata în cadrul formei evenimentul *KeyUp* (este necesară folosirea acestui eveniment deoarece evenimentul *KeyPressed* nu permite determinarea tastelor speciale) pentru a intercepta şi formele alternative (CTRL+INS, SHIFT+DEL, …). Pentru ca forma să aibă posibilitatea de a intercepta mesajele de la tastatură indiferent de ce control are focus-ul trebuie setată proprietatea *KeyPreview* pe *true*. Codul pentru tratarea evenimentului este:

```
private void FormaSpectacole_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
        if (e.Shift && e.KeyCode == Keys.Delete)
        {
                Decupare();
                e.Handled = true;
        }

        if (e.Control && e.KeyCode == Keys.Insert)
        {
                Copiere();
                e.Handled = true;
        }

        if (e.Shift && e.KeyCode == Keys.Insert)
        {
                Lipire();
                e.Handled = true;
        }
}
```

Pentru implementarea operațiilor au fost create două funcții ajutătoare (una pentru adăugarea elementului în clipboard şi una pentru desenarea unui spectacol într-un bitmap):

```
void PuneElement()  // adauga spectacolul curent in clipboard
{
        // obtinem o referinta la spectacolul selectat
        Spectacol spectacol = (Spectacol)lbSpectacole.SelectedItem;

        // construim varianta text (compatibil cu Microsoft Excel)
        string strSpectacol = string.Format(
                "{0}\t{1}\t{2}", spectacol.Denumire,
                spectacol.OraInceput, spectacol.OraSfarsit);

        // construim varianta grafica (similara cu cea de la DrawItem)
        Bitmap imgSpectacol = DesenareSpectacol(spectacol);

        // construim obiectul care va fi pus in clipboard
        DataObject date = new DataObject();
```

```
        date.SetData(typeof(string), strSpectacol);
        date.SetData(typeof(Bitmap), imgSpectacol);

        Clipboard.SetDataObject(date);
}

Bitmap DesenareSpectacol(Spectacol s)
{
        // construim imaginea
        Bitmap imagine = new Bitmap(DimZonaFixa * 2, lbSpectacole.ItemHeight);

        // construim referinta la suprafata pe care vom desena
        Graphics g = Graphics.FromImage(imagine);

        // construim pensulele
        Brush bText   = Brushes.Black;
        Brush bFundal = Brushes.White;

        // desenam fundalul
        GraphicsUnit pixel = GraphicsUnit.Pixel;
        RectangleF limite = imagine.GetBounds(ref pixel);
        g.FillRectangle(bFundal, limite);

        // ----- Desenare denumire spectacol ----- //

        // construim fontul cu care vom desena denumirea
        Font fDenumire = new Font(Font, FontStyle.Bold);

        // zona in care desenam
        Rectangle rectDenumire = new Rectangle(
                (int)limite.Left, (int)limite.Top,
                (int)limite.Width - DimZonaFixa - 3,
                (int)limite.Height);

        // formatarea pentru denumire
        StringFormat formatDenumire =  new StringFormat();
        formatDenumire.Alignment     = StringAlignment.Near;
        formatDenumire.LineAlignment = StringAlignment.Center;
        formatDenumire.FormatFlags = StringFormatFlags.NoWrap;
        formatDenumire.Trimming = StringTrimming.EllipsisCharacter;

        // desenam denumirea
        g.DrawString(s.Denumire, fDenumire, bText, rectDenumire, formatDenumire);

        // ----- Desenare text ore ----- //

        // zona in care desenam
        RectangleF rectTextOre = new RectangleF(
                limite.Right - DimZonaFixa,
                limite.Top + 2,
                DimZonaFixa - 6, 12);

        // formatarea pentru sfarsit (la dreapta)
        StringFormat formatSfarsit =  new StringFormat();
        formatSfarsit.Alignment     = StringAlignment.Far;

        // desenam orele
        g.DrawString("Inceput: " + s.OraInceput, Font, bText, rectTextOre);
        g.DrawString("Sfarsit: " + s.OraSfarsit, Font, bText, rectTextOre, formatSfarsit);

        // ----- Desenare grafica ore ----- //

        int start = (int)limite.Right - DimZonaFixa;

        // desenam fundalul alb
        g.FillRectangle(Brushes.White, start, limite.Bottom - 15, 6*24, 10);

        // desenam zona rosie
        g.FillRectangle(Brushes.Red,
                start+ s.OraInceput * 6,
                limite.Bottom - 15,
                (s.OraSfarsit - s.OraInceput) * 6, 10);

        // desenam gradatiile
        for (int i = 0; i < 24; i++)
                g.DrawLine(Pens.Black,
                        start + i * 6,
```

```
                        limite.Bottom - 15,
                        start + i * 6,
                        limite.Bottom - 5);

        // desenam conturul
        g.DrawRectangle(Pens.Black, start, limite.Bottom - 15, 145, 10);

        // ne asiguram ca toate operatiile au fost executate
        g.Flush(FlushIntention.Sync);

        return imagine;
}
```

Funcțiile care implementează operațiile cu clipboard-ul sunt următoarele:

```
void Decupare()
{
        // trebuie sa avem un spectacol selectat
        if (lbSpectacole.SelectedItem == null)
                return;

        // punem elementul in clipboard
        PuneElement();

        // si stergem elementul fara sa intrebam
        StergeSpectacol(false);
}

void Copiere()
{
        // trebuie sa avem un spectacol selectat
        if (lbSpectacole.SelectedItem == null)
                return;

        // punem elementul in clipboard
        PuneElement();
}

void Lipire()
{
        // scoatem datele din clipboard
        IDataObject date = Clipboard.GetDataObject();

        // daca sunt de tip text
        if (date.GetDataPresent(typeof(string)))
        {
                // scoatem datele
                string strDate = (string)date.GetData(typeof(string));

                string[] linii = strDate.Split('\r');
                foreach(string linie in linii)
                        // incercam sa convertim linia intr-un spectacol
                        try
                        {
                                Spectacol s = new Spectacol(
                                        linie.Split('\t')[0],
                                        int.Parse(linie.Split('\t')[1]),
                                        int.Parse(linie.Split('\t')[2]));

                                // daca am reusit conversia adaugam spectacolul
                                Model.AdaugaSpectacol(s);
                        }
                        // ignoram erorile aparute
                        catch {}
        }

}
```

Implementarea pentru drag and drop se face similar, in trei etape:

1. La *MouseDown* se crează obiectul şi se iniţializează operaţia
2. La *DragEnter* se verifică dacă datele sunt într-un format compatibil
3. La *DragDrop* se extrag datele

Codul care implementează aceste operații este:

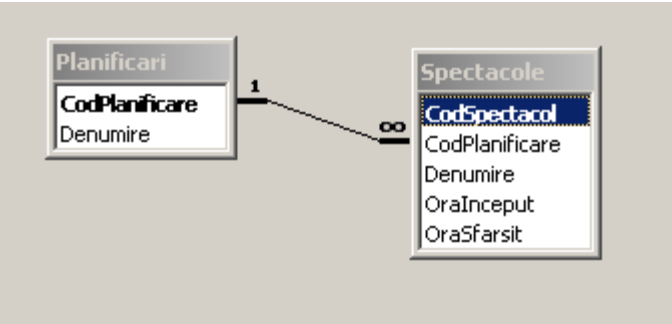
```
private void lbSpectacole_MouseDown(object sender, System.Windows.Forms.MouseEventArgs e)
{
        // obtinem pozitia elementului
        int index = lbSpectacole.IndexFromPoint(e.X,  e.Y);

        if (index >= 0)
        {
                // obtinem o referinta la spectacolul selectat
                Spectacol spectacol = (Spectacol)lbSpectacole.Items[index];

                // construim varianta text (compatibil cu Microsoft Excel)
                string strSpectacol = string.Format(
                        "{0}\t{1}\t{2}", spectacol.Denumire,
                        spectacol.OraInceput, spectacol.OraSfarsit);

                // construim varianta grafica (similara cu cea de la DrawItem)
                Bitmap imgSpectacol = DesenareSpectacol(spectacol);

                // construim obiectul care va fi pus in clipboard
                DataObject date = new DataObject();
                date.SetData(typeof(string), strSpectacol);
                date.SetData(typeof(Bitmap), imgSpectacol);
                date.SetData(typeof(Spectacol), spectacol.Clone());

                lbSpectacole.DoDragDrop(date, DragDropEffects.Copy);
        }
}

private void lbSpectacole_DragEnter(object sender, System.Windows.Forms.DragEventArgs e)
{
        // intial consideram ca nu este permis
        e.Effect = DragDropEffects.None;

        if (e.Data.GetDataPresent(typeof(Spectacol)))
        {
                // scoatem datele
                Spectacol spectacol = (Spectacol)e.Data.GetData(typeof(Spectacol));

                e.Effect = DragDropEffects.Copy;

                // verificam daca nu exista
                foreach(Spectacol s in Model)
                        if (s.ToString() == spectacol.ToString())
                                e.Effect = DragDropEffects.None;
        }

        if (e.Data.GetDataPresent(typeof(string)))
        {
                // scoatem datele
                string strDate = (string)e.Data.GetData(typeof(string));

                string[] linii = strDate.Split('\r');
                foreach(string linie in linii)
                        // incercam sa convertim linia intr-un spectacol
                        try
                        {
                                Spectacol spectacol = new Spectacol(
                                        linie.Split('\t')[0],
                                        int.Parse(linie.Split('\t')[1]),
                                        int.Parse(linie.Split('\t')[2]));

                                // daca am reusit cel putin o conversie
                                // permitem actiunea
                                e.Effect = DragDropEffects.Copy;

                                // verificam daca nu exista
                                foreach(Spectacol s in Model)
                                        if (s.ToString() == spectacol.ToString())
                                                e.Effect = DragDropEffects.None;

                        }
                                // ignoram erorile aparute
```

```
                    catch {}
        }
}

private void lbSpectacole_DragDrop(object sender, System.Windows.Forms.DragEventArgs e)
{
        if (e.Data.GetDataPresent(typeof(Spectacol)))
        {
                // extragem referinta la spectacol
                Spectacol s = (Spectacol)e.Data.GetData(typeof(Spectacol));

                // si adaugam o copie in modelul nostru
                Model.AdaugaSpectacol((Spectacol)s.Clone());

                return;
        }

        if (e.Data.GetDataPresent(typeof(string)))
        {
                // scoatem datele
                string strDate = (string)e.Data.GetData(typeof(string));

                string[] linii = strDate.Split('\r');
                foreach(string linie in linii)
                        // incercam sa convertim linia intr-un spectacol
                        try
                        {
                                Spectacol s = new Spectacol(
                                        linie.Split('\t')[0],
                                        int.Parse(linie.Split('\t')[1]),
                                        int.Parse(linie.Split('\t')[2]));

                                // daca am reusit conversia adaugam spectacolul
                                Model.AdaugaSpectacol(s);

                        }
                        // ignoram erorile aparute
                        catch {}
        }
}
```

Versiunea completă este disponibilă aici.

Versiunea 5: Utilizare baze de date

Pentru stocarea datelor s-a folosit o bază de date Access cu următoarea structură:



Șirul de conectare la baza de date va fi reținut într-un fișier de configurare (Add → New Item → Application Configuration File) cu următorul conținut:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
        <appSettings>
            <add
                key="SirConectare"
```

```
                value="Provider=Microsoft.Jet.OLEDB.4.0;Data
                Source=D:\ASE\POO\!Aplicatie\Spectacole\Spectacole.mdb;Mode=Share Deny
                None;"/>
        </appSettings>
</configuration>
```

Şirul de conectare poate fi construit vizual folosind View → Server Explorer din Visual Studio după care Data Connections → Add Connection.

Pentru selectarea planificării care trebuie încărcată s-a construit o fereastră de dialog numită *FormaDeschideDb*. Populareа listei s-a făcut cu ajutorul mecanismului de data binding:

```
lbPlanificari.DataSource    = ListaPlanificari();
lbPlanificari.DisplayMember = "Denumire";
lbPlanificari.ValueMember   = "CodPlanificare";
```

Obținerea tabelei cu lista planificărilor este implementată în funcția:

```
private DataTable ListaPlanificari()
{
        // construim o tabela noua
        DataTable listaPlanificari = new DataTable("Planificari");

        // construim conexiunea
        OleDbConnection dbConn = new
OleDbConnection(ConfigurationSettings.AppSettings["SirConectare"]);
        try
        {
                dbConn.Open();  // deschidem conexiunea la BD

                // folosim un adaptor pentru popularea tabelei din memorie
                OleDbDataAdapter dbAdapter = new OleDbDataAdapter("SELECT * FROM
Planificari", dbConn);
                dbAdapter.Fill(listaPlanificari);
        }
        finally
        {
                // inchidem conexiunea
                dbConn.Close();
        }

        return listaPlanificari;
}
```

Încărcarea efectivă a datelor în model este realizată folosind un DataReader si un obiect de tip Command în forma principala:

```
private void DeschideDB()
{
      // atentionam utilizatorul ca va pierde modificarile daca continua
      if (Modificat)
      {
              DialogResult rezultat = MessageBox.Show(this,
                      "Doriti sa incarcati o planificare noua si sa pierdeti modificarile?",
                      "Planificare Spectacole", MessageBoxButtons.YesNo,
      MessageBoxIcon.Warning);

              // utilizatorul refuza atunci anulam operatia
              if (rezultat == DialogResult.No)
                      return;
      }

      // obtinem codul planificarii
      FormaDeschideDB forma = new FormaDeschideDB();
      forma.StartPosition = FormStartPosition.CenterParent;

      if (forma.ShowDialog(this) == DialogResult.OK)
      {
```

```
            Model.StergeLista();   // stergem spectacolele

            // daca userul nu a anulat comanda
            // citim datele din baza de date
            OleDbConnection dbConn = new
      OleDbConnection(ConfigurationSettings.AppSettings["SirConectare"]);
            try
            {
                  dbConn.Open(); // deschidem conexiunea la baza de date

                  // construim comanda
                  OleDbCommand dbCmd = new OleDbCommand("SELECT * FROM SPECTACOLE WHERE
      CodPlanificare = ?", dbConn);
                  dbCmd.Parameters.Add(new OleDbParameter("?", forma.CodPlanificare));

                  // obtinem datele
                  OleDbDataReader dbReader = dbCmd.ExecuteReader();

                  // adaugam spectacolele in model
                  while(dbReader.Read())
                        Model.AdaugaSpectacol(new Spectacol(
                              dbReader.GetString(2),dbReader.GetInt32(3),
      dbReader.GetInt32(4)));
                  dbReader.Close();

                  AtasareEvenimente();

                  Modificat = false;     // si resetam indicatorul de modificat
                  NumeFisier = "[DB]";
            }
            finally
            {
                  // inchidem conexiunea
                  dbConn.Close();
            }
      }
}
```

Salvarea datelor este realizată folosind obiecte de tip Command:

```
public void SalvareDB()
{
      FormaSalvareDB forma = new FormaSalvareDB();
      forma.StartPosition = FormStartPosition.CenterParent;

      if (forma.ShowDialog(this) == DialogResult.OK)
      {
            // daca userul nu a anulat comanda
            // citim datele din baza de date
            OleDbConnection dbConn = new
      OleDbConnection(ConfigurationSettings.AppSettings["SirConectare"]);
            try
            {
                  dbConn.Open(); // deschidem conexiunea la baza de date

                  // adaugam inregistrarea in planificari
                  OleDbCommand dbCmdPlanif = new OleDbCommand("INSERT INTO Planificari
      (Denumire) VALUES(?)", dbConn);
                  dbCmdPlanif.Parameters.Add(new OleDbParameter("?",
      forma.NumePlanificare));
                  dbCmdPlanif.ExecuteNonQuery();

                  // obtinem codul planificarii
                  OleDbCommand dbCmdCod = new OleDbCommand("SELECT MAX(CodPlanificare) FROM
      Planificari", dbConn);
                  int codPlanificare = (int)dbCmdCod.ExecuteScalar();

                  // adaugam spectacolele
                  OleDbCommand dbCmdSpect = new OleDbCommand("INSERT INTO Spectacole
      (CodPlanificare, Denumire, OraInceput, OraSfarsit) VALUES(?, ?, ?, ?)", dbConn);
                  dbCmdSpect.Parameters.Add(new OleDbParameter("?", DbType.Int32));
                  dbCmdSpect.Parameters.Add(new OleDbParameter("?", DbType.String));
                  dbCmdSpect.Parameters.Add(new OleDbParameter("?", DbType.Int32));
                  dbCmdSpect.Parameters.Add(new OleDbParameter("?", DbType.Int32));
```

```
                foreach(Spectacol spectacol in Model)
                {
                        dbCmdSpect.Parameters[0].Value = codPlanificare;
                        dbCmdSpect.Parameters[1].Value = spectacol.Denumire;
                        dbCmdSpect.Parameters[2].Value = spectacol.OraInceput;
                        dbCmdSpect.Parameters[3].Value = spectacol.OraSfarsit;


                        dbCmdSpect.ExecuteNonQuery();
                }

                AtasareEvenimente();

                Modificat = false;            // resetam indicatorul
            }
            finally
            {
                // inchidem conexiunea
                dbConn.Close();
            }
        }
}
```

Versiunea completă este disponibilă aici.