Medii vizuale de programare
Curs 2

Conf. dr.ing. GENGE Béla

Universitatea "Petru Maior", Departamentul de Informatică
Tîrgu Mureș, România
bela.genge@ing.upm.ro

Clase și instanțe în C#

- Declararea unei clase
  - Constructori
  - Destructori
  - Modificatori de acces (se pot combina dacă nu sunt opuse): **public**, **internal**, **protected**, **private**, **sealed**
  - Metode de clasă și de instanță
- Pentru clase din același namespace se aplică doar **public, internal** și **sealed**. Pentru clase imbricate se aplică și celelalte.

Exemplu

```
class FormaGeom {
  public FormaGeom() {...}
  ~FormaGeom() {
  }
  public int aria() {
   return 0;
  }
}
```

Moștenirea

- Apelul constructorilor și destructorilor

Exemplu 1

```
class FormaGeom {
  public FormaGeom() {
  }
  ~FormaGeom() {
  }
}
```

Exemplu 2

```
class Cerc : FormaGeom {
  public Cerc() : base() {
  }
  ~Cerc() {
  }
}
```

Moștenirea

- Apelul constructorilor și destructorilor

Exemplu 1

```
class FormaGeom {
  public FormaGeom() {
  }
  ~FormaGeom() {
  }
}
```

Exemplu 2

```
class Cerc : FormaGeom {
  public Cerc() : base() {
  }
  ~Cerc() {
  }
}
```

Supradefinirea metodelor

- Supradefinirea metodei `aria`
- `virtual`, `override`
- Instanțiere prin clasa de bază: `FormaGeom fg = new Cerc();`
- Instanțiere prin clasa derivată: `Cerc fg = new Cerc();`
- Apelul explicit al metodelor din clasa de bază: `base.denumireMetodă()`

Exemplu

```
class FormaGeom {
...
  public virtual int aria() {
    ... }
}
class Cerc : FormaGeom {
  public override int aria() {
    ... }
}
```

Variabile membru (câmpuri)

- Se aplică modificatorii de acces anteriori
- Variabile membru de instanță și de clasă
- Declararea constantelor membru: `const` și `readonly`
- Scrierea și citirea variabilelor membru prin metode

Exemplu

```
class Cerc : FormaGeom {
  private double raza = 0;
  protected readonly double PI = Math.PI;
  public static double afisare = true;
...
}
```

Proprietăți

- Proprietăți get și set

Exemplu

```
public double Raza {
  get
  {
   return raza;
  }
  set
  {
   raza = value;
  }
}
Cerc c = new Cerc();
double xx = c.Raza;
c.Raza = y;
```

Interfețe

- Cuvântul cheie `interface`
- Definesc un șablon de metode ce trebuie implementate
- Nu pot conține declarații de variabile
- Nu se pot aplica modificatori de acces

Exemplu

```
interface IFormeGeom {
  int raza();
}
class FormaGeom : IFormeGeom {
  public int raza() {
    ...
  }
}
```

Clase abstracte

- Cuvântul cheie `abstract`
- O clasă este abstractă dacă conține cel puțin o metodă abstractă
- Clasele abstracte nu pot fi instanțiate
- Clasa derivată trebuie să implementeze metodele abstracte

Exemplu

```
abstract class FormeGeom {
  ...
  public abstract int raza();
}
```

Exercițiu

- Crearea unei clase singleton.

Excepții

- Reprezintă o metodă elegantă pentru tratarea erorilor
- Presupune identificarea și separarea instrucțiunilor generatoare de excepții
- Cuvinte cheie: `try`, `catch` și `finally`

Exemplu

```
try {
  int nr = int.Parse(Console.ReadLine()); }
catch (Exception ex) {
  Console.WriteLine("EXCEPTIE: " + ex.ToString()); {
finally {
  Console.WriteLine("Bloc finally"); }
```

Delegarea metodelor

- C# pune la dispoziție tipul `delegate` pentru a facilita transferul metodelor ca și parametrii
- Reprezintă echivalentul pointerilor către funcții din 'C'
- Orice metodă ce corespunde semnăturii (prototipului) unui delegat poate fi apelată prin obiectul delegat

Exemplu

```
public delegate int sumaNumere(int a, int b);
```

Fire de execuție

- Sunt cunoscute sub denumirea de procese ușoare - nu sunt procese
- Un set de instrucțiuni folosit pentru izolarea unui task
- Clasa `Thread` din `System.Threading`
- Se folosește tipul delegat `ThreadStart` pentru crearea firelor

Exemplu

```
Thread th = new Thread(new ThreadStart(myth.run));
...
  private void run() {
   while (true) {
    Console.WriteLine("Code is running in own thread");
   }
  }
```

Metode uzuale

- `Start()`: pornire fire de execuție
- `IsAlive`: testare execuție
- `Thread.Sleep(ms)`: deplanificare în milisecunde
- `Abort()`: cerere oprire fir de execuție
- `Join()`: așteptare oprire

- Metodă deprecated: `Suspend()`: suspendarea firului de execuție
- Metodă deprecated: `Resume()`: reluarea execuției
- Problema: nu se cunoaște momentul suspendării execuției

Sincronizarea accesului la resurse comune

- Metode cea mai simplă: definirea unei secțiuni critice prin `lock`
- Sincronizarea se face pe un anumit obiect
- A se evita sincronizarea pe instanțe publice (e.g., this)

Exemplu

```
private volatile object _sync_pause = new object();
...
  // Blocare execuție
  lock(_sync_pause) {
...
  }
```

Sincronizarea prin clasa `Monitor`

- Notificarea eliberării unei resurse, i.e., lock
- Clasa `Monitor` nu se instanțiază
- Metode de interes: `Enter` și `Exit`, `Wait` și `Pulse`
- Exemplu: implementarea metodelor deprecate `Suspend` și `Resume`
  - Pasul 1: fără `Monitor`
  - Pasul 2: cu `Monitor`

Sincronizarea prin clasa Monitor

```
class MyThread
{
    static private readonly object _o = new object();
    private Thread _th;
    public MyThread()
    {
        _th = new Thread(new ThreadStart(this.run));
        _th.Start();
    }
```

Sincronizarea prin clasa Monitor

```
private void run()
{
    Console.WriteLine("Thread: started");
    Thread.Sleep(1);

    // Exemplu 1
    Monitor.Enter(_o);
    Console.WriteLine("Thread: entered ex1");
    Thread.Sleep(1000);
    Monitor.Exit(_o);
    Console.WriteLine("Thread: exited ex1");

    // Exemplu 2
    Console.WriteLine("Thread: entered ex2");
    lock (_o)
    {
        Monitor.Wait(_o);
    }
    Console.WriteLine("Thread: exited ex2");
    Thread.Sleep(1000);
    Console.WriteLine("Thread: ended");
}
```

Sincronizarea prin clasa Monitor

```
public void continueThread()
{
    // Exemplu 2
    lock (_o)
    {
        Monitor.PulseAll(_o);
    }
}
public void waitThread()
{
    _th.Join();
}
```

Sincronizarea prin clasa Monitor

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Main: App started");
        MyThread th = new MyThread();
        Thread.Sleep(5000);

        Console.WriteLine("Main: Calling the thread pulse");
        th.continueThread();

        Console.WriteLine("Main: Calling the thread wait");
        th.waitThread();
```

Thread pools

- Un set de fire de execuție administrate automat de .NET.
- Clasa `ThreadPool`.
- Metode de interes: `GetMinThreads`, `GetMaxThreads`, `GetAvailableThreads` și `ThreadPool.QueueUserWorkItem`

```
public static void Main()
{
    // Queue the task.
    ThreadPool.QueueUserWorkItem(ThreadProc);
    Console.WriteLine("Main thread does some work, then sleeps.");
    Thread.Sleep(1000);

    Console.WriteLine("Main thread exits.");
}

// This thread procedure performs the task.
static void ThreadProc(Object stateInfo)
{
    // No state object was passed to QueueUserWorkItem, so stateInfo is null.
    Console.WriteLine("Hello from the thread pool.");
}
```

Clase parțial-definite

- C# permite definirea aceleiași clase în mai multe fișiere
- Cuvântul cheie `partial`

Exemplu

```
partial class ClasaP {
  public void m1() {...}
}
partial class ClasaP {
  public void m2() {...}
}
```

Aplicație: text în mișcare pe o fereastră

- Se vor folosi fire de execuție
- Crearea firului în constructorul clasei `Form1`
- Distrugerea nu este posibilă în destructor - nu stim cand se apelează de GC (se va implementa o metodă apelată din `Dispose`)
- Varianta 2: implementarea unei clase distincte MyThread (constructorul va primi un delegat către metoda ce realizează desenarea pe fereastră).
- Se va folosi:
  - `System.Drawing.SolidBrush(Color.Red);`
  - `System.Drawing.Font("Arial", 16);`
  - `Graphics g = this.CreateGraphics();`
  - `g.DrawString();`
  - În final: `obj.Dispose();` pentru fiecare obiect grafic

Erori și Debugging

- Tratarea erorilor face parte din ciclul de viață a oricărei aplicații
- Tipuri de erori:
  - Erori de sintaxă: tratare la compilare
  - Erori la execuție: tratare prin excepții
  - Erori logice: dificil de tratat, necesită debugging

Erori de sintaxă

- Relativ ușor de identificat
- Sunt evidențiate în faza de preprocesare/compilare

Exemplu

```
using System.Thread; //în loc de using System.Threading;
```

Erori la execuție

- Pot fi identificate/anticipate de programatori
- Depind de intrările recepționate sau de contextul execuției
- Aplicația va eșua în anumite contexte

Exemplu - utilizatorul introduce un tip de dată greșit

```
int i = int.Parse(Console.ReadLine());
// Soluție: bloc try-catch (System.FormatException)
```

Erori logice

- Cel mai dificil de identificat
- Programul se compilează și rulează fără probleme
- Rezultatele returnate sunt neașteptate

Exemplu - utilizarea greșită a operatorilor

```
int i = 0;
int j = i++;//Se vroia pre-incrementare
```

Debugging

- Compilare Release și Debug
- Permite execuția linie cu linie
- Activare debugger: step into/over, atingere breakpoint, sau excepție

- Lansare Debug: F5
- Stop Debug: Shift + F5
- Exemplu: ciclu for pentru afisarea mesajelor cu contor
- Vizualizare: Autos, Locals, Watch

Debugging - exemplu monitorizare și break fire de execuție

- Crearea unui fir de execuție
- Lansare debugger (F5)
- Lansare Break (Debug->Break All)
- Vizualizare fire de execuție (Debug->Windows->Threads)

Fișiere de logare

Scrierea într-un fișier

```
StreamWriter _fs = null;
try
{
  if (File.Exists(fName))
  {
    File.Delete(fName);
  }
  _fs = new StreamWriter(fName);
}
catch (Exception ex)
{
  throw ex;
}
```

Fișiere de logare

Construirea mesajelor de logare

```
string logMsg = DateTime.Now.ToString("MM\\/dd\\/yyyy
h\\:mm:ss tt");
logMsg += " - ";
logMsg += msg;
```

Metode necesare pentru clasa MessageLogger

```
getInstance();
initialize(fName);
close();
logMessage(msg);
```

Clasele Debug și Trace

- C# pune la dispoziție clase specializate pentru verificarea condițiilor, logarea și afișarea mesajelor
- Clasele `Debug` și `Trace` nu se instanțiază
- Definite în spațiul de nume `System.Diagnostics`

- Clasa `Debug`:
  - Folosită în faza de dezvoltare a aplicației
  - Asigură un set de metode pentru testarea condițiilor, afișarea și salvarea mesajelor
- Clasa `Trace`:
  - Folosită în faza de deployment (release) a aplicației
  - Asigură un set de metode pentru monitorizarea și diagnoza aplicației

Clasele Debug și Trace - exemple

- `Debug.Write` și `Debug.WriteLine`
- `Trace.Write` și `Trace.WriteLine`
- `Debug.WriteIf` și `Debug.WriteLineIf`
- `Trace.WriteIf` și `Trace.WriteLineIf`
- `Debug.Fail` și `Trace.Assert`

Trace listeners

- Ieșirile metodelor din clasele `Debug` și `Trace` pot fi redirecționate
- Implicit, ieșirea este direcționată către debugger
- Cei mai importanți listeneri:
  - `TextWriterTraceListener`: logarea mesajelor sub forma de fișiere text
  - `EventLogTraceListener`: logarea mesajelor în lista de evenimente Windows

Trace listeners - exemplu TextWriterTraceListener

Exemplu

```
System.IO.FileStream f = new
      System.IO.FileStream("testLog.txt",
                  System.IO.FileMode.OpenOrCreate);
TextWriterTraceListener txtLst = new
      TextWriterTraceListener(f);
Trace.Listeners.Add(txtLst);
Trace.AutoFlush = true;
Trace.WriteLine("Mesaj de log");
```

Configurarea comutatoarelor Trace

- Tracing-ul poate fi activat/dezactivat dintr-un fișier XML exterior
- Click dreapta pe denumirea proiectului → Add → New Item → Application Configuration File

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <switches>
      <add name="Switch1" value="1" />
      <add name="Switch2" value="0" />
    </switches>
  </system.diagnostics>
</configuration>
```

Exemplu utilizare

- Clasele `BooleanSwitch` și `TraceSwitch`
- `BooleanSwitch`: **On** (value=1) și **Off** (value=0)
- `TraceSwitch`: **Off** (value=0), **Error** (value=1), **Warning** (value=2), **Info** (value=3), **Verbose** (value=4)

Exemplu

```
BooleanSwitch sw1 = new BooleanSwitch("Switch1", "Text
Control Bool");
TraceSwitch tr1 = new TraceSwitch("Switch2", "Text Control
Trace");
Debug.WriteIf(sw1.Enabled, "Mesaj debug");
```