



Ministerul Educație,
Cercetări și Tineretului

Microsoft®

Introducere în Programarea .Net Framework



Autori

Grup elaborare suport curs:

M.E.C.T.:

Nușa Dumitriu-Lupan, Inspector General M.E.C.T.
Rodica Pintea, profesor, Liceul Grigore Moisil, București
Adrian Niță, profesor, Colegiul Național Emanuil Gojdu, Oradea
Mioara Niță, profesor, Colegiul Național Emanuil Gojdu, Oradea
Cristina Sichim, profesor, Colegiul Național Ferdinand I, Bacău
Nicolae Olăroiu, profesor Colegiul Național "B.P. Hașdeu", Buzău

MICROSOFT:

Mihai Tătărani, cadru didactic asociat, Universitatea Politehnica Timișoara
Petru Jucovschi, Developer Community Lead, Microsoft România
Tudor-Ioan Salomie, Team Lead Microsoft Student Partners,
Universitatea Tehnică Cluj Napoca

**Programarea
Orientată pe Obiecte
și
Programarea Vizuală
cu C# .Net**

CUPRINS

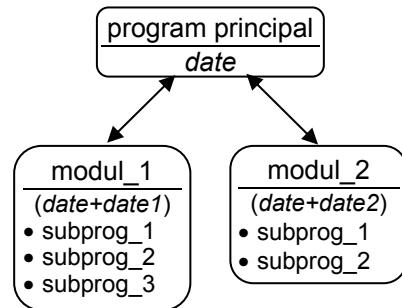
CUPRINS.....	1
1. PROGRAMAREA ORIENTATĂ OBIECT (POO)	3
1.1. EVOLUȚIA TEHNICILOR DE PROGRAMARE.....	3
1.2. TIPURI DE DATE OBIECTUALE. ÎNCAPSULARE	3
1.3. SUPRAÎNCĂRCARE	4
1.4. MOȘTENIRE	5
1.5. POLIMORFISM. METODE VIRTUALE	5
1.6. PROGRAMARE ORIENTATĂ OBIECT ÎN C#.....	6
1.7. DECLARAREA UNEI CLASE	6
1.8. CONSTRUCTORI	7
1.9. DESTRUCTOR.....	7
1.10. METODE	8
1.11. PROPRIETĂȚI	9
1.12. EVENIMENTE ȘI DELEGĂRI.....	10
1.13. INTERFEȚE.....	11
1.14. FIRE DE EXECUȚIE	12
2. PLATFORMA .NET	13
2.1. PREZENTARE	13
2.2. .NET FRAMEWORK	13
2.3. COMPILEAREA PROGRAMELOR	14
3. LIMBAJUL C#.....	14
3.1. CARACTERIZARE	14
3.2. COMPILEAREA LA LINIA DE COMANDĂ	14
3.3. CREAREA APLICAȚIILOR CONSOLĂ	15
3.4. STRUCTURA UNUI PROGRAM C#	16
3.5. SINTAXA LIMBAJULUI.....	16
3.6. TIPURI DE DATE	17
3.7. CONVERSII.....	21
3.8. CONSTANTE.....	22
3.9. VARIABILE	22
3.10. EXPRESII ȘI OPERATORI	22
3.11. COLECȚII	23
3.12. INSTRUCȚUNEA FOREACH	23
3.13. INSTRUCȚIUNILE TRY-CATCH-FINALLY ȘI THROW	23
4. PROGRAMARE VIZUALĂ	25
4.1. CONCEPTE DE BAZĂ ALE PROGRAMĂRII VIZUALE	25
4.2. MEDIUL DE DEZVOLTARE VISUAL C#.....	26
4.3. FERESTRE	27
4.4. CONTROALE	29
4.5. SYSTEM.DRAWING	36
4.6. VALIDAREA INFORMAȚIILOR DE LA UTILIZATOR	37
5. APLICAȚII ORIENTATE PE DATE.....	38
5.1. STRUCTURI DE DATE.....	38
5.2. COLECȚII DE DATE.....	38
5.3. ADO.NET	39
5.4. CONECTAREA LA O SURSĂ DE DATE.....	39
5.5. EXECUTAREA UNEI COMENZI SQL	41
5.6. SETURI DE DATE	42
5.7. PROIECTAREA VIZUALĂ A SETURILOR DE DATE	43

1. Programarea Orientată Obiect (POO)

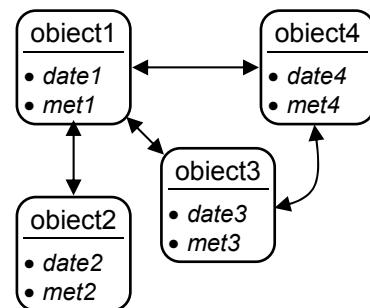
1.1. Evoluția tehniciilor de programare

- **Programarea nestructurată** (un program simplu, ce utilizează numai variabile globale); complicațiile apar când prelucrarea devine mai amplă, iar datele se multiplică și se diversifică.
- **Programarea procedurală** (program principal deservit de subprograme cu parametri formali, variabile locale și apeluri cu parametri efectivi); se obțin avantaje privind depanarea și reutilizarea codului și se aplică noi tehnici privind transferul parametrilor și vizibilitatea variabilelor; complicațiile apar atunci când la program sunt asignați doi sau mai mulți programatori care nu pot lucra simultan pe un același fișier ce conține codul sursă.
- **Programarea modulară** (gruparea subprogramelor cu funcționalități similare în module, implementate și depanate separat); se obțin avantaje privind independenta și **încapsularea** (prin separarea zonei de implementare, păstrând vizibilitatea numai asupra zonei de interfață a modulului) și se aplică tehnici de asociere a procedurilor cu datele pe care le manevrează, stabilind și diferite reguli de acces la date și la subprograme.

Se observă că modulele sunt "centrate" pe proceduri, acestea gestionând și setul de date pe care le prelucrează (*date+date₁* din figură). Dacă, de exemplu, dorim să avem mai multe seturi diferențiate de date, toate înzestrate comportamental cu procedurile din modulul *modul_1*, această arhitectură de aplicație nu este avantajoasă.



- **Programarea orientată obiect** (programe cu noi *tipuri* ce integrează atât datele, cât și metodele asociate creării, prelucrării și distrugerii acestor date); se obțin avantaje prin **abstractizarea** programării (programul nu mai este o succesiune de prelucrări, ci un ansamblu de obiecte care prind viață, au diverse proprietăți, sunt capabile de acțiuni specifice și care interacționează în cadrul programului); intervin tehnici noi privind instantierea, derivarea și polimorfismul tipurilor obiectuale.



1.2. Tipuri de date obiectuale. Încapsulare

Un tip de date abstract (ADT) este o entitate caracterizată printr-o *structură de date* și un *ansamblu de operații* aplicabile acestor date. Considerând, în rezolvarea unei probleme de gestiune a accesului utilizatorilor la un anumit site, tipul abstract *USER*, vom observa că sunt multe date ce caracterizează un utilizator Internet. Totuși se va ține cont doar de datele semnificative pentru problema dată. Astfel, "culoarea ochilor" este irelevantă în acest caz, în timp ce "data nașterii" poate fi importantă. În aceeași idee, operații specifice ca "se înregistrează", 'comandă on-line" pot fi relevante, în timp ce operația "manâncă" nu este, în cazul nostru. Evident, nici nu se pun în discuție date sau operații nespecifice ("numărul de laturi" sau acțiunea "zboară").

Operațiile care sunt accesibile din afara entității formează *interfața* acesteia. Astfel, operații interne cum ar fi conversia datei de naștere la un număr standard calculat de la 01.01.1900 nu fac parte din interfața tipului de date abstract, în timp ce operația "plasează o comandă on-line" face parte, deoarece permite interacțiunea cu alte obiecte (SITE, STOC etc.)

O *instanță* a unui tip de date abstract este o "concretizare" a tipului respectiv, formată din valori efective ale datelor.

Un *tip de date obiectual* este un tip de date care implementează un tip de date abstract. Vom numi operațiile implementate în cadrul tipului de date abstract *metode*. Spunem că datele și metodele sunt *membrii* unui tip de date obiectual. Folosirea unui astfel de tip presupune: existența definiției acestuia, apelul metodelor și accesul la date.

Un exemplu de-acum clasic de tip de date abstract este STIVA. Ea poate avea ca date: numerele naturale din stivă, capacitatea stivei, vârful etc. Iar operațiile specifice pot fi: introducerea în stivă (*push*) și extragerea din stivă (*pop*). La implementarea tipului STIVA, vom defini o structură de date care să rețină valorile memorate în stivă și câmpuri de date simple pentru: capacitate, număr de elemente etc. Vom mai defini metode (subprograme) capabile să creeze o stivă vidă, care să introducă o valoare în stivă, să extragă valoarea din vârful stivei, să testeze dacă stiva este vidă sau dacă stiva este plină etc.

Crearea unei instanțe noi a unui tip obiectual, presupune operațiuni specifice de "construire" a noului obiect, metoda corespunzătoare purtând numele de *constructor*. Analog, la desființarea unei instanțe și eliberarea spațiului de memorie aferent datelor sale, se aplică o metodă specifică numită *destructor*¹.

O aplicație ce utilizează tipul obiectual STIVA, va putea construi două sau mai multe stive (de cărți de joc, de exemplu), le va umple cu valori distincte, va muta valori dintr-o stivă în alta după o anumită regulă desființând orice stivă golită, până ce rămâne o singură stivă. De observat că toate aceste prelucrări recurg la datele, constructorul, destrutorul și la metodele din interfața tipului STIVA descris mai sus.

Principalul tip obiectual întâlnit în majoritatea mediilor de dezvoltare (Visual Basic, Delphi, C++, Java, C#) poartă numele de clasă (*class*). Există și alte tipuri obiectuale (*struct*, *object*). O instanță a unui tip obiectual poartă numele de *obiect*.

La implementare, datele și metodele asociate trebuie să fie complet și corect definite, astfel încât utilizatorul să nu fie nevoie să țină cont de detalii ale acestei implementări. El va accesa datele, prin intermediul proprietăților și va efectua operațiile, prin intermediul metodelor puse la dispoziție de tipul obiectual definit. Spunem că tipurile de date obiectuale respectă principiul *încapsulării*. Astfel, programatorul ce utilizează un tip obiectual CONT (în bancă) nu trebuie să poarte grija modului cum sunt reprezentate în memorie datele referitoare la un cont sau a algoritmului prin care se realizează actualizarea soldului conform operațiilor de depunere, extragere și aplicare a dobânzilor. El va utiliza unul sau mai multe conturi (instanțe ale tipului CONT), accesând proprietățile și metodele din interfață, realizatorul tipului obiectual asumându-și acele griji în momentul definirii tipului CONT.

Permitând extensia tipurilor de date abstracte, clasele pot avea la implementare:

- date și metode caracteristice fiecărui obiect din clasă (membri de tip instanță),
- date și metode specifice clasei (membri de tip clasă).

Astfel, clasa STIVA poate beneficia, în plus, și de date ale clasei cum ar fi: numărul de stive generate, numărul maxim sau numărul minim de componente ale stivelor existente etc. Modificatorul static plasat la definirea unui membru al clasei face ca acela să fie un membru de clasă, nu unul de tip instanță. Dacă în cazul membrilor nestatici, există câte un exemplar al membrului respectiv pentru fiecare instanță a clasei, membrii statici sunt unici, fiind accesăți în comun de toate instanțele clasei. Mai mult, membrii statici pot fi referiți fără a crea vreo instanță a clasei respective.

1.3. Supraîncărcare

Deși nu este o tehnică specifică programării orientată obiect, ea creează un anumit context pentru metodele ce formează o clasă și modul în care acestea pot fi (ca orice subprogram) apelate.

Prin supraîncarcare se înțelege posibilitatea de a defini în același domeniu de vizibilitate² mai multe funcții cu același nume, dar cu parametri diferiti ca tip și/sau ca număr. Astfel ansamblul format din numele funcției și lista sa de parametri reprezintă o modalitate unică de identificare numită *semnătură* sau amprentă. Supraîncărcarea permite obținerea unor efecte diferite ale apelului în contexte diferite³.

¹ Datorită tehnicii de supraîncărcare C++, Java și C# permit existența mai multor constructori

² Notiunile generale legate de vizibilitate se consideră cunoscute din programarea procedurală. Aspectele specifice și modificatorii de acces/vizibilitate pot fi studiați din documentațiile de referință C#.

³ Capacitatea unor limbaje (este și cazul limbajului C#) de a folosi ca "nume" al unui subprogram un operator, reprezintă supraîncărcarea operatorilor. Aceasta este o facilitate care

Apelul unei funcții care beneficiază, prin supraîncărcare, de două sau mai multe semnături se realizează prin selecția funcției a cărei semnătură se potrivește cel mai bine cu lista de parametri efectivi (de la apel).

Astfel, poate fi definită metoda "comandă on-line" cu trei semnături diferite:

`comanda_online(cod_prod)` cu un parametru întreg (desemnând comanda unui singur produs identificat prin `cod_prod`).

`comanda_online(cod_prod,cantitate)` cu primul parametru întreg și celalalt real

`comanda_online(cod_prod,calitate)` cu primul parametru întreg și al-II-lea caracter.

1.4. Moștenire

Pentru tipurile de date obiectuale `class` este posibilă o operație de extindere sau specializare a comportamentului unei clase existente prin definirea unei clase noi ce moștenește datele și metodele clasei de bază, cu această ocazie putând fi redefiniți unii membri existenți sau adăugați unii membri noi. Operația mai poartă numele de *derivare*.

Clasa din care se moștenește se mai numește *clasa de bază* sau *superclasă*. Clasa care moștenește se numește *subclasă*, clasă derivată sau clasă *descendentă*.

Ca și în Java, în C# o subclasă poate moșteni de la o singură superclasă, adică avem de-a face cu moștenire simplă; aceeași superclasă însă poate fi derivată în mai multe subclase distincte. O subclasă, la randul ei, poate fi superclasă pentru o altă clasă derivată. O clasă de bază impreună cu toate clasele descendente (direct sau indirect) formează o ierarhie de clase. În C#, toate clasele moștenesc de la clasa de bază `Object`.

În contextul mecanismelor de moștenire trebuie amintiți modificatorii `abstract` și `sealed` aplicări unei clase, modificatori ce obligă la și respectiv se opun procesului de derivare. Astfel, o clasă abstractă trebuie obligatoriu derivată, deoarece direct din ea nu se pot obține obiecte prin operația de instantiere, în timp ce o clasă sigilată (`sealed`) nu mai poate fi derivată (e un fel de terminal în ierarhia claselor). O metodă abstractă este o metodă pentru care nu este definită o implementare, aceasta urmând să fie realizată în clasele derive din clasa curentă⁴. O metodă sigilată nu mai poate fi redefinită în clasele derive din clasa curentă.

1.5. Polimorfism. Metode virtuale

Folosind o extensie a sensului etimologic, un obiect polimorfic este cel capabil să ia diferite forme, să se afle în diferite stări, să aibă comportamente diferite. Polimorfismul obiectual⁵ se manifestă în lucrul cu obiecte din clase aparținând unei ierarhii de clase, unde, prin redefinirea unor date sau metode, se obțin membri diferenți având însă același nume. Astfel, în cazul unei referiri obiectuale, se pune problema stabilirii datei sau metodei referite. Comportamentul polimorfic este un element de flexibilitate care permite stabilirea contextuală, în mod dinamic⁶, a membrului referit.

De exemplu, dacă este definită clasa numită `PIESA` (de săh), cu metoda nestatică `muta(pozitie_initiala,pozitie_finala)`, atunci subclasele `TURN` și `PION` trebuie să aibă metoda `muta` definită în mod diferit (pentru a implementa maniera specifică a pionului de a captura o piesă "en passant"⁷). Atunci, pentru un obiect `T`, aparținând claselor

"reduce" diferențele dintre operarea la nivel abstract (cu DTA) și apelul metodei ce realizează acestă operație la nivel de implementare obiectuală. Deși ajută la sporirea expresivității codului, prin supraîncărcarea operatorilor și metodelor se pot crea și confuzii.

⁴ care trebuie să fie și ea abstractă (virtuală pură, conform terminologiei din C++)

⁵ deoarece tot aspecte polimorfice îmbracă și unele tehnici din programarea clasică sau tehnica supraîncărcării funcțiilor și operatorilor.

⁶ Este posibil doar în cazul limbajelor ce permit "legarea întârziată". La limbajele cu "legare timpurie", adresa la care se face un apel al unui subprogram se stabilește la compilare. La limbajele cu legare întârziată, această adresa se stabilește doar în momentul rulării, putându-se calcula distinct, în funcție de contextul în care apare apelul.

⁷ Într-o altă concepție, metoda `muta` poate fi implementată la nivelul clasei `PIESA` și redefinită la nivelul subclasei `PION`, pentru a particulariza acest tip de deplasare care capturează piesa peste care trece pionul în diagonală.

derivate din PIESA, referirea la metoda mută pare nedefinită. Totuși mecanismele POO permit stabilirea, în momentul apelului, a clasei proxime căreia îi aparține obiectul T și apelarea metodei corespunzătoare (mutare de pion sau tură sau altă piesă).

Pentru a permite acest mecanism, metodele care necesită o decizie contextuală (în momentul apelului), se decală ca metode virtuale (cu modificadorul `virtual`). În mod curent, în C# modificadorul `virtual` al funcției din clasa de bază, îi corespunde un specificator `override` al funcției din clasa derivată ce redefineste funcția din clasa de bază.

O metodă ne-virtuală nu este polimorfică și, indiferent de clasa căreia îi aparține obiectul, va fi invocată metoda din clasa de bază.

1.6. Programare orientată obiect în C#

C# permite utilizarea OOP respectând toate principiile enunțate anterior.

Toate componentele limbajului sunt într-un fel sau altul, asociate noțiunii de clasă. Programul însuși este o clasă având metoda statică `Main()` ca punct de intrare, clasă ce nu se instantiază. Chiar și tipurile predefinite `byte`, `int` sau `bool` sunt clase sigilate derivate din clasa `ValueType` din spațiul `System`. Pentru a evita unele tehnici de programare periculoase, limbajul oferă tipuri speciale cum ar fi: interfețe și delegări. Versiunii 2.0 a limbajului i s-a adăugat un nou tip: clasele generice⁸,

1.7. Declararea unei clase

Sintaxa⁹: [atrib]o [modificatori]o **class** [nume_clasă] [:clasa_de_bază]o [corp_clasă].

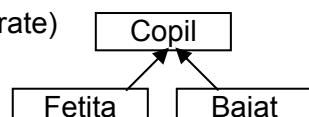
Atributele reprezintă informații declarative cu privire la entitatea definită.

Modificatorii reprezintă o secvență de cuvinte cheie dintre: `new public protected internal private` (modificatori de acces) `abstract sealed` (modificatori de moștenire). Clasa de bază este clasa de la care moștenește clasa curentă și poate exista o singură astfel de clasă de bază. Corpul clasei este un bloc de declarări ale membrilor clasei: `constante` (valori asociate clasei), `câmpuri` (variabile), `tipuri` de date definite de utilizator, `metode` (subprograme), `constructori`, un `destructor`, `proprietăți` (caracteristici ce pot fi consultate sau setate), `evenimente` (instrumente de semnalizare), `indexatori` (ce permit indexarea instanțelor din cadrul clasei respective) și operatori.

- constructorii și destructorul au ca nume numele clasei proxime din care fac parte¹⁰
- metodele au nume care nu coincid cu numele clasei sau al altor membri (cu excepția metodelor, conform mecanismului de supraîncărcare)
- metodele sau constructorii care au același nume trebuie să difere prin semnătură¹¹
- se pot defini date și metode statice (caracteristice clasei) și un constructor static care se execută la inițializarea clasei propriu-zise; ele formează un fel de "context" al clasei
- se pot defini date și metode nestatice (de instanță) care se multiplică pentru fiecare instanță în parte în cadrul operației de instanțiere; ele formează contextele tuturor instanțelor clasei respective

Exemplul următor definește o ierarhie de clase (conform figurii alăturate)

```
public abstract class Copil
public class Fetita: Copil { }
public sealed class Baiat: Copil { }
```



Modificadorul `abstract` este folosit pentru a desemna faptul că nu se pot obține obiecte din clasa `Copil`, ci numai din derivelele acesteia (`Fetita`, `Baiat`), iar modificadorul `sealed` a fost folosit pentru a desemna faptul că nu se mai pot obține clase derive din clasa `Baiat` (de exemplu, subclasele `Baiat_cuminte` și `Baiat_rau`)

⁸ echivalentul claselor `template` din C++

⁹ `[]` din definiția schematică semnifică un neterminat, iar `_o` semnifică o componentă optională

¹⁰ având în vedere că ele pot să facă parte dintr-o clasă interioară altiei clase

¹¹ din semnătură nefăcând parte specificatorii `ref` și `out` asociați parametrilor

1.8. Constructori

Sintaxă:

[atrib]o [modificatori]o [nume_clasă] ([listă_param_formali]o) [:inițializator]o [corp_constr]o

Modificatori: public protected internal private extern

Inițializator: **base**([listă_param]o), **this**([listă_param]o) ce permite invocarea unui constructor anume¹² înainte de executarea instrucțiunilor ce formează corpul constructorului curent. Dacă nu este precizat niciun inițializator, se asociază implicit inițializatorul **base**().

Corpul constructorului este format din instrucțiuni care se execută la crearea unui nou obiect al clasei respective (sau la crearea clasei, în cazul constructorilor cu modificadorul static).

- pot exista mai mulți constructori care se pot diferenția prin lista lor de parametri
- constructorii nu pot fi moșteniți
- dacă o clasă nu are definit niciun constructor, se va asigna automat constructorul fără parametri al clasei de bază (clasa object, dacă nu este precizată clasa de bază)

Instanțierea presupune declararea unei variabile de tipul clasei respective și inițializarea acesteia prin apelul constructorului clasei (unul dintre ei, dacă sunt definiți mai mulți) precedat de operatorul new. Acestea se pot realiza și simultan într-o instrucțiune de felul:

[Nume_clasă] [nume_object]=new [Nume_clasă] ([listă_param]o)

➤ Utilizarea unui constructor fără parametri și a constructorului implicit în clasă derivată

```
public abstract class Copil
{ protected string nume;
  public Copil() {nume = Console.ReadLine();} //la inițializarea obiectului se citește
                                                //de la tastatură un sir de caractere ce va reprezenta numele copilului
}
class Fetita:Copil {}
```

```
...
Fetita f=new Fetita();
Copil c= new Copil();           //Pentru clasa Copil abstractă, s-ar fi obținut eroare aici
```

➤ Suprăîncărcarea constructorilor și definirea explicită a constructorilor în clase derivate

```
public class Copil
{ protected string nume; //dată accesibilă numai în interiorul clasei și claselor derivate
  public Copil() {nume = Console.ReadLine();}
  public Copil(string s) {nume=s;}
}
class Fetita:Copil
{ public Fetita(string s):base(s) {nume="Fetita "+nume}13
  public Fetita(){}
  //public Fetita(string s):base() {nume=s}
}
...
```

```
Copil c1= new Copil();           //se citeste numele de la tastatură
Copil c2= new Copil("Codrina");
Fetita f1=new Fetita();Fetita f2=new Fetita("Ioana");
```

Există două motive pentru care definiția constructorului al treilea din clasa Fetita este greșită și de aceea este comentată. Care sunt aceste motive?

1.9. Destructor

Sintaxă: [atrib]o [extern]o ~[nume_clasă] () [corp_destructor]o

Corpul destructorului este format din instrucțiuni care se execută la distrugerea unui obiect al clasei respective. Pentru orice clasă poate fi definit un singur constructor. Destructorii

¹² Din clasa de bază (base) sau din clasa insăși (this)

¹³ Preia și specializează constructorul al doilea din clasa de bază

¹⁴ Este echivalent cu public Fetita():base(){}

nu pot fi moșteniți. În mod normal, destructorul nu este apelat în mod explicit, deoarece procesul de distrugere a unui obiect este invocat și gestionat automat de Garbage Collector.

1.10. Metode

Sintaxă: [atrib]o [modificatori]o [tip_returnat] [nume] ([listă_param_formali]o) [corp_metoda]o

Modificatori: new public protected internal private static virtual abstract sealed override extern¹⁵

Tipul rezultat poate fi un tip definit sau void. Numele poate fi un simplu identificator sau, în cazul în care definește în mod explicit un membru al unei interfețe, numele este de forma [nume_interfata].[nume_metoda]

Lista de parametri formali este o succesiune de declarări despărțite prin virgule, declararea unui parametru având sintaxa: [atrib]o [modificatori]o [tip] [nume]

Modificatorul unui parametru poate fi ref (parametru de intrare și ieșire) sau out (parametru care este numai de ieșire). Parametrii care nu au niciun modificador sunt parametri de intrare.

Un parametru formal special este parametrul tablou cu sintaxa: [atrib]o params [tip] I I [nume].

- Pentru metodele abstracte și externe, corpul metodei se reduce la un semn ;
 - Semnătura fiecărei metode este formată din numele metodei, modificatorii acesteia, numărul și tipul parametrilor¹⁶
 - Numele metodei trebuie să difere de numele oricărui alt membru care nu este metodă.
 - La apelul metodei, orice parametru trebuie să aibă același modificador ca la definire
- Invocarea unei metode se realizează prin sintagma [nume_object].[nume_metoda] (pentru metodele nestatice) și respectiv [nume_clasă].[nume_metoda] (pentru metodele statice).

➤ Definirea datelor și metodelor **statice** corespunzătoare unei clase

```
public class Copil
{
    public const int nr_max = 5;                                //constantă
    public static int nr_copii=0;                               //câmp simplu (variabilă)
    static Copil[] copii=new Copil[nr_max];                  //câmp de tip tablou (variabilă)
    public static void adaug_copil(Copil c)                   //metodă
    {
        copii[nr_copii++]= c;
        if (nr_copii==nr_max) throw new Exception("Prea multi copii");
    }
    public static void afisare()                                //metodă
    {
        Console.WriteLine("Sunt {0} copii:", nr_copii);
        for (int i = 0; i<nr_copii; i++)
            Console.WriteLine("Nr.{0}. {1}", i+1, copii[i].nume);
    } ...17
}
...
Fetita c = new Fetita();Copil.adaug_copil(c);
referința nouului obiect se memorează în tabloul static copii (caracteristic clasei) și se incrementează data statică nr_copii
Baiat c = new Baiat(); Copil.adaug_copil(c);
Copil c = new Copil(); Copil.adaug_copil(c);
Copil.afisare(); //se afișează o listă cu numele celor 3 copii
```

¹⁵ Poate fi folosit cel mult unul dintre modificatorii static, virtual și override ; nu pot apărea împreună new și override, abstract nu poate să apară cu niciunul dintre static, virtual, sealed, extern; private nu poate să apară cu niciunul dintre virtual, override și abstract; seald obligă și la override

¹⁶ Din semnătură (amprentă) nu fac parte tipul returnat, numele parametrilor formali și nici specifiicatorii ref și out.

¹⁷ Se are în vedere și constructorul fără parametri definit și preluat implicit în subclasele din cadrul primului exemplu din subcapitolul 1.8: public Copil() {nume = Console.ReadLine();}

➤ Definirea datelor și metodelor **nestatiche** corespunzătoare clasei Copil și claselor derivate

```

public class Copil
{
    ...
    public string nume;
    public virtual void se_joaca()      //virtual → se poate suprascrie la derivare
        {Console.WriteLine("{0} se joaca.", this.nume);}
    public void se_joaca(string jucaria)      //nu permite redefinire18
        {Console.WriteLine("{0} se joaca cu {1}.", this.nume, jucaria);}
    }
class Fetita:Copil
{
    public override void se_joaca()      //redefinire → comportament polimorfic
        {Console.WriteLine("{0} leagana papusa.",this.nume);}
}
class Baiat:Copil
{
    public override void se_joaca()
        {Console.WriteLine("{0} chinuie pisica.",this.nume);}
}
...
Fetita c = new Fetita();c.se_joaca("pisica");c.se_joaca();      //polimorfism
Baiat c = new Baiat();c.se_joaca("calculatorul");c.se_joaca(); //polimorfism
Copil c = new Copil();c.se_joaca();      //polimorfism

```

Pentru a evidenția mai bine comportamentul polimorfic, propunem secvența următoare în care nu se știe exact ce este obiectul copii[i] (de tip Copil, Fetita sau Baiat?):

```
for (int i=0; i<nr_copii; i++) copii[i].se_joaca;
```

1.11. Proprietăți

Proprietatea este un membru ce permite accesul controlat la datele-membri ale clasei.

Sintaxa: [atrib]_o [modificator]_o [tip] [nume_proprietate] {[metode_de_acces]}_o

Observațiile privind modificatorii și numele metodelor sunt valabile și în cazul proprietăților.

Metodele de acces sunt două: **set** și **get**. Dacă proprietatea nu este abstractă sau externă, poate să apară una singură dintre cele două metode de acces sau amândouă, în orice ordine. Este o manieră de lucru recomandabilă aceea de a proteja datele membru (câmpuri) ale clasei, definind instrumente de acces la acestea: pentru a obține valoarea câmpului respectiv (**get**) sau de a memora o anumită valoare în câmpul respectiv (**set**). Dacă metoda de acces **get** este perfect asimilabilă cu o metodă ce returnează o valoare (valoarea datei pe care vrem să o obținem sau valoarea ei modificată conform unei prelucrări suplimentare specifice problemei în cauză), metoda **set** este asimilabilă cu o metodă care un parametru de tip valoare (de intrare) și care atribuie (sau nu, în funcție de context) valoarea respectivă câmpului. Cum parametrul corespunzător valorii transmise nu apare în structura sintactică a metodei, este de stiuț că el este implicit identificat prin cuvântul **value**. Dacă se supune unor condiții specifice problemei, se face o atribuire de felul **câmp=value**.

➤ Definirea în clasa Copil a proprietății Nume, corespunzătoare câmpului protejat ce reține, sub forma unui sir de caractere, numele copilului respectiv. Se va observă că proprietatea este moștenită și de clasele derivate Fetita și Baiat¹⁹.

```

public class Copil
{
    ...
    string nume; // este implicit protected
    public string Nume //proprietatea Nume
    {
        get
        {
            if(char.ToUpper(nume[0]))return nume; else return nume.ToUpper();
        }
        set { nume = value; }
    }
}
```

¹⁸ Decât cu ajutorul modificadorului **new** pentru metoda respectivă în clasa derivată

¹⁹ Desigur că proprietatea care controlează accesul la câmpul identificat prin nume se poate numi cu totul altfel (proprietatea Nume fiind ușor de confundat cu câmpul de date nume).

```

    public Copil() {Nume = Console.ReadLine();}                                //metoda set
}
class Fetita:Copil
{   public override void se_joaca()
    {Console.WriteLine("{0} leagana papusa.",this.Nume);}                      //metoda get
}20

```

1.12. Evenimente și delegări

Evenimentele sunt membri ai unei clase ce permit clasei sau obiectelor clasei să facă notificări, adică să anunțe celelalte obiecte asupra unor schimbări petrecute la nivelul stării lor. Clasa furnizoare a unui eveniment *publică* (pune la dispoziția altor clase) acest lucru printr-o declarare *event* care asociază evenimentului un *delegat*, adică o referință către o funcție necunoscută căreia î se precizează doar antetul, funcția urmând a fi implementată la nivelul claselor interesate de evenimentul respectiv. Este modul prin care se realizează comunicarea între obiecte. Tehnica prin care clasele implementează metode (*handler-e*) ce răspund la evenimente generate de alte clase poartă numele de *tratare a evenimentelor*.

Sintaxa: [atrib]o[modificatori]o **event** [tip_delegat] [nume]

Modificatorii permisi sunt aceiași ca și la metode.

Tipul delegat este un tip de date ca oricare altul, derivat din clasa sigilată *Delegate*, din spațiul *System*. Definirea unui tip delegat se realizează prin declararea:

[atrib]o[modificatori]o **delegate** [tip_rezultat] [nume_delegat] ([listă_param_formali]o)

Un delegat se poate defini și în afara clasei generatoare de evenimente și poate servi și altor scopuri în afara tratării evenimentelor. Prezentăm în continuare un exemplu.

De exemplu, dacă dorim să definim o metodă asociată unui vector de numere întregi, metodă ce verifică dacă vectorul este o succesiune "bine aranjată" (orice două valori successive respectă o anumită regulă), o implementare "generică" se poate realiza folosind delegări:

```

public delegate bool pereche_ok(object t1, object t2);

public class Vector
{
    public const int nmax = 4;
    public int[] v=new int[nmax];
    public Vector()
    {
        Random rand = new Random();
        for (int i = 0; i < nmax; i++) v[i] = rand.Next(0,5);
    }
    public void scrie()
    {
        for (int i = 0; i < nmax; i++) Console.Write("{0}, ", v[i]);
        Console.WriteLine();
    }
    public bool aranj(pereche_ok ok)//ok e o delegare către o funcție necunoscută
    {
        for (int i = 0; i < nmax-1; i++)
            if (!ok(v[i], v[i + 1])) return false;
        return true;
    }
}

```

Dacă în clasa-program²¹ se adaugă funcțiile (exprimând două "reguli de aranjare" posibile)

```

public static bool f1(object t1, object t2)
{
    if ((int)t1 >= (int)t2) return true;else return false;
}
public static bool f2(object t1, object t2)
{
    if ((int)t1 <= (int)t2) return true;else return false;
}

```

atunci o secvență de prelucrare aplicativă ar putea fi:

²⁰ De observat că în exemplul anterior (subcapitolul 1.10), câmpul nume era declarat public, pentru a permite accesul "general" la câmpul respectiv de date. Iar metodele și constructorii foloseau identificatorul nume și nu proprietatea Nume.

²¹ Independent de definiția clasei Vector

```

static void Main(string[] args)
{
    Vector x;
    do {
        x = new Vector(); x.scrie();
        if (x.aranj(f1)) Console.WriteLine("Monoton descrescator");
        if (x.aranj(f2)) Console.WriteLine("Monoton crescator");
    } while (Console.ReadKey(true).KeyChar != '\x001B'); //Escape
}

```

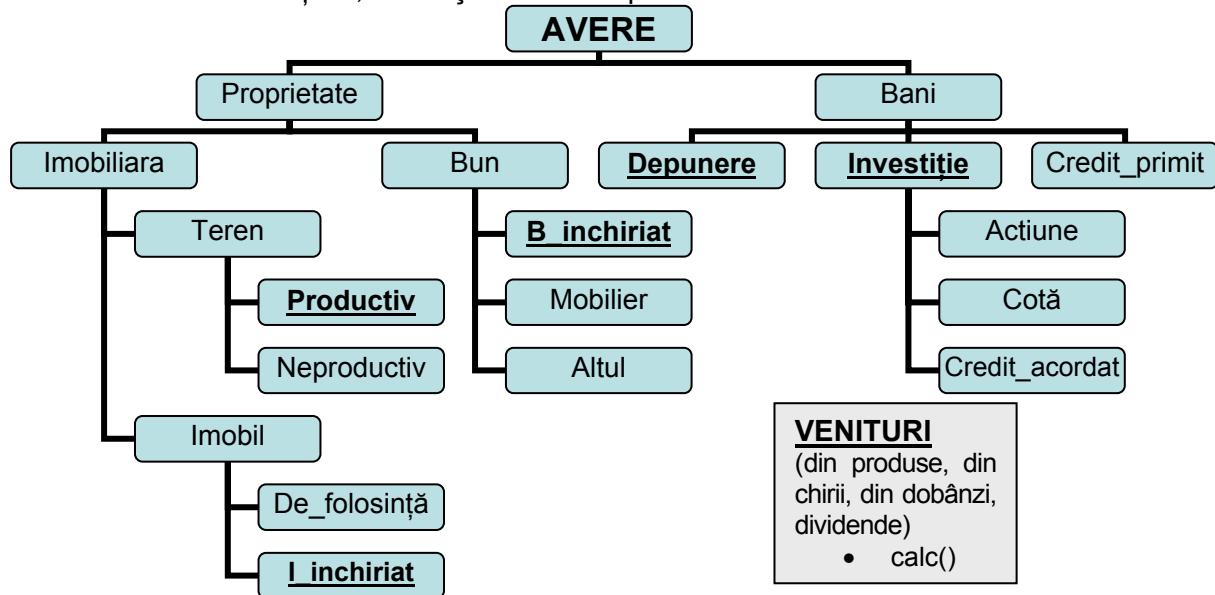
Revenind la evenimente, descriem pe scurt un exemplu teoretic de declarare și tratare a unui eveniment. În clasa `Vector` se consideră că interschimbarea valorilor a două componente ale unui vector e un eveniment de interes pentru alte obiecte sau clase ale aplicației. Se definește un tip delegat `TD` (să zicem) cu niște parametri de interes²² și un eveniment care are ca asociat un delegat `E` (de tip `TD`)²³. Orice obiect `x` din clasa `Vector` are un membru `E` (înțial `null`). O clasă `C` interesată să fie înștiințată când se face vreo interschimbare într-un vector pentru a genera o animație (de exemplu), va implementa o metodă `M` ce realizează animația și va adăuga pe `M` (prin intermediul unui delegat) la `x.E`²⁴. Cumulând mai multe astfel de referințe, `x.E` ajunge un fel de listă de metode (*handlere*). În clasa `Vector`, în metoda `sort`, la interschimbarea valorilor a două componente se invocă delegatul `E`. Invocarea lui `E` realizează de fapt activearea tuturor metodelor adăugate la `E`.

Care credeți că sunt motivele pentru care apelăm la evenimente în acest caz, când pare mult mai simplu să apelăm direct metoda `M` la orice interschimbare?

1.13. Interfețe

Interfețele sunt foarte importante în programarea orientată pe obiecte, deoarece permit utilizarea polimorfismului într-un sens mai extins. O interfață este o componentă a aplicației, asemănătoare unei clase, ce declară prin membrii săi (metode, proprietăți, evenimente și indexatori) un "comportament" unitar aplicabil mai multor clase, comportament care nu se poate defini prin ierarhia de clase a aplicației.

De exemplu, dacă vom considera arborele din figura următoare, în care `AVERE` este o clasă abstractă, iar derivarea claselor a fost concepută urmărind proprietățile comune ale componentelor unei averi, atunci o clasă `VENIT` nu este posibilă, deoarece ea ar moșteni de la toate clasele evidențiate, iar moștenirea multiplă nu este admisă în C#.



²² De exemplu indicii componentelor interschimbate

²³ A se observa că evenimentul în sine este anonim, doar delegatul asociat are nume

²⁴ Într-o atribuire de felul `x.E+=new [tip_delegat](M)`

Pentru metodele din cadrul unei interfețe nu se dă nici o implementare, ci sunt pur și simplu specificate, implementarea lor fiind furnizată de unele dintre clasele aplicației²⁵. Nu există instantiere în cazul interfețelor, dar se admit derivări, inclusiv moșteniri multiple.

În exemplul nostru, se poate defini o interfață VENIT care să conțină antetul unei metode calc (să zicem) pentru calculul venitului obținut, fiecare dintre clasele care implementează interfața VENIT fiind obligată să furnizeze o implementare (după o formulă de calcul specifică) pentru metoda calc din interfață. Orice clasă care dorește să adere la interfață trebuie să implementeze toate metodele din interfață. Toate clasele care moștenesc dintr-o clasă care implementează o interfață moștenesc, evident, metodele respective, dar le pot și redifini (de exemplu, clasa Credit_acordat redefinește metoda calc din clasa Investiție, deoarece formula de calcul implementată acolo nu își poate "potrivește" și ei²⁶).

De exemplu, dacă presupunem că toate clasele subliniate implementează interfața VENIT, atunci pentru o avere cu actiuni la două firme, un imobil închiriat și o depunere la bancă, putem determina venitul total:

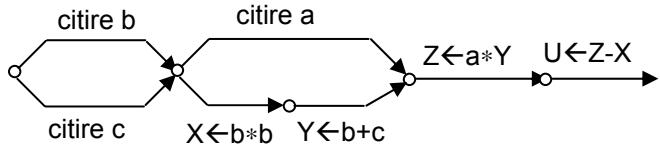
```
Actiune act1 = new Actiune(); Actiune act2 = new Actiune();
I_inchiriat casa = new I_inchiriat(); Depunere dep=new Depunere();
Venit[] venituri = new Venit()[4];
venituri[0] = act1; venituri[1] = act2;
venituri[2] = casa; venituri[3] = dep;
...
int t=0;
for(i=0;i<4;i++) t+=v[i].calc();
```

Găsiți două motive pentru care interfața VENIT și rezolvarea de mai sus oferă o soluție mai bună decât: $t=act1.calc() + act2.calc() + casa.calc() + dep.calc()$.

1.14. Fire de execuție

Programarea prelucrărilor unei aplicații pe mai multe fire de execuție (*multithreading*) presupune "descompunerea" ansamblului de prelucrări în secvențe, unele dintre ele "planificându-se" și prelucrate paralel (cvasi-simultan) de către procesor în vederea utilizării eficiente a acestuia.

Schema alăturată este o posibilă planificare a procesului de calcul al expresiei $a*(b+c)-b*b$, unde a, b și c sunt niște matrice pătrate ale căror componente se găsesc în fișiere separate.



În orice punct al prelucrării de pe un fir de execuție se poate genera un nou fir (*thread*) căruia îi se asociază un subprogram cu ajutorul unui delegat. În exemplul următor, care utilizează spațiul *System.Threading*, afișarea succesivă a unor secvențe de 1 și de 2 demonstrează "comutarea" executării de pe un fir pe altul la intervale relativ constante de timp:

```
static void scrie2()
{
    for (int i = 1; i <= 300; i++) Console.Write('2');
}
static void Main(string[] args)
{
    ThreadStart delegat = new ThreadStart(scrie2);
    Thread fir = new Thread(delegat);
    fir.Start();
    for (int i = 1; i <= 500; i++) Console.Write('1');
    Console.ReadKey();
}
```

Spațiul Threading mai oferă facilități de schimbare a priorităților prelucrărilor din fire paralele, de delimitare a secvențelor critice, de așteptare și semnalizare folosind semafoare etc. O clasă sigilată folosită curent în aplicații (*Timer*) permite implementarea unui ceas care funcționează pe un fir paralel cu aplicația curentă și care, la intervale de timp ce se pot seta, generează un eveniment (*Tick*) "sesizat" și exploatat de prelucrarea din firul principal.

²⁵ Acele clase care "aderă" la o interfață spunem că "implementează" interfața respectivă

²⁶ Dacă în sens polimorfic spunem că Investiție este și de tip Bani și de tip Avere, tot așa putem spune că o clasă care implementează interfața VENIT și clasele derivate din ea sunt și de tip VENIT

2. Platforma .NET

2.1. Prezentare

.NET este un cadru (*framework*) de dezvoltare software unitară care permite realizarea, distribuirea și rularea aplicațiilor-desktop Windows și aplicațiilor WEB.

Tehnologia .NET pune laolaltă mai multe tehnologii (ASP, XML, OOP, SOAP, WDSL, UDDI) și limbaje de programare (VB, C++, C#, J#) asigurând totodată atât portabilitatea codului compilat între diferite calculatoare cu sistem Windows, cât și reutilizarea codului în programe, indiferent de limbajul de programare utilizat.

.NET Framework este o componentă livrată împreună cu sistemul de operare Windows. De fapt, .NET 2.0 vine cu Windows Server 2003 și Windows XP SP2 și se poate instala pe versiunile anterioare, până la Windows 98 inclusiv; .NET 3.0 vine instalat pe Windows Vista și poate fi instalat pe versiunile Windows XP cu SP2 și Windows Server 2003 cu minimum SP1.

Pentru a dezvolta aplicații pe platforma .NET este bine să avem 3 componente esențiale:

- un set de limbaje (C#, Visual Basic .NET, J#, Managed C++, Smalltalk, Perl, Fortran, Cobol, Lisp, Pascal etc),
- un set de medii de dezvoltare (Visual Studio .NET, Visio),
- și o bibliotecă de clase pentru crearea serviciilor Web, aplicațiilor Web și aplicațiilor desktop Windows.

Când dezvoltăm aplicații .NET, putem utiliza:

- Servere specializate - un set de servere Enterprise .NET (din familia SQL Server 2000, Exchange 2000 etc), care pun la dispoziție funcții de stocare a bazelor de date, email, aplicații B2B (*Business to Business* – comerț electronic între partenerii unei afaceri).
- Servicii Web (în special comerciale), utile în aplicații care necesită identificarea utilizatorilor (de exemplu, .NET Passport - un mod de autentificare folosind un singur nume și o parolă pentru toate site-urile vizitate)
- Servicii incluse pentru dispozitive non-PC (Pocket PC Phone Edition, Smartphone, Tablet PC, Smart Display, XBox, set-top boxes, etc.)

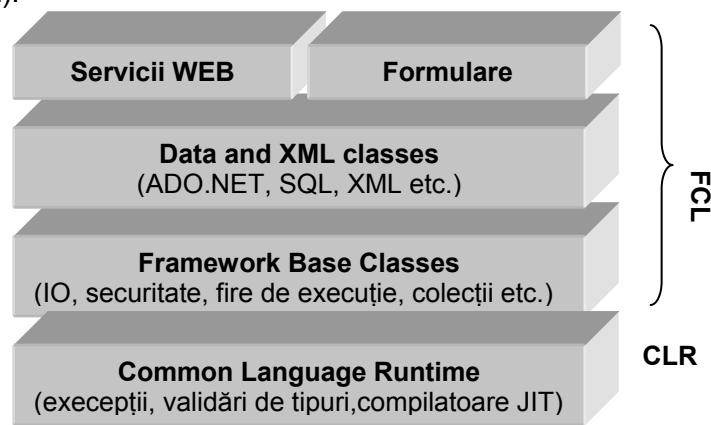
2.2. .NET Framework

Componenta .NET Framework stă la baza tehnologiei .NET, este ultima interfață între aplicațiile .NET și sistemul de operare și actualmente conține:

- Limbajele C#, VB.NET, C++ și J#. Pentru a fi integrate în platforma .NET toate aceste limbaje respectă niște specificații OOP numite *Common Type System* (CTS). Ele au ca elemente de bază: clase, interfețe, delegări, tipuri valoare și referință, iar ca mecanisme: moștenire, polimorfism și tratarea excepțiilor.
- Platforma comună de executare a programelor numită *Common Language Runtime* (CLR), utilizată de toate cele 4 limbaje.
- Ansamblul de biblioteci necesare în realizarea aplicațiilor desktop sau Web numit *Framework Class Library* (FCL).

Arhitectura .NET Framework

Componenta .NET Framework este formată din compilatoare, biblioteci și alte executabile utile în rularea aplicațiilor .NET. Fișierele corespunzătoare se află, în general, în directorul WINDOWS\Microsoft.NET\Framework\V2.0.... (corespunzător versiunii instalate)



2.3. Compilarea programelor

Un program scris într-unul dintre limbajele .NET conform *Common Language Specification* (CLS) este compilat în *Microsoft Intermediate Language* (MSIL sau IL). Codul astfel obținut are extensia *exe*, dar nu este direct executabil, ci respectă formatul unic MSIL.

CLR include o mașină virtuală asemănătoare cu o mașină Java, ce execută instrucțiunile IL rezultate în urma compilării. Mașina folosește un compilator special JIT (*Just In Time*). Compilatorul JIT analizează codul IL corespunzător apelului unei metode și produce codul mașină adecvat și eficient. El recunoaște secvențele de cod pentru care s-a obținut deja codul mașină adecvat permitând reutilizarea acestuia fără recomplire, ceea ce face ca, pe parcursul rulării, aplicațiile .NET să fie din ce în ce mai rapide.

Faptul că programul IL produs de diferitele limbaje este foarte asemănător are ca rezultat interoperabilitatea între aceste limbaje. Astfel, clasele și obiectele create într-un limbaj specific .NET pot fi utilizate cu succes într-un program scris în alt limbaj.

În plus, CLR se ocupă de gestionarea automată a memoriei (un mecanism implementat în platforma .NET fiind acela de eliberare automată a zonelor de memorie asociate unor date devenite inutile – *Garbage Collection*).

Ca un element de portabilitate, trebuie spus că CTS are o arhitectură ce permite rularea aplicațiilor .NET, în afară de Windows, și pe unele tipuri de Unix, Linux, Solaris, Mac OS X și alte sisteme de operare (http://www.mono-project.com/Main_Page).

3. Limbajul C#

3.1. Caracterizare

Limbajul C# fost dezvoltat de o echipă restrânsă de ingineri de la Microsoft, echipă din care s-a evidențiat Anders Hejlsberg (autorul limbajului Turbo Pascal și membru al echipei care a proiectat Borland Delphi).

C# este un limbaj simplu, cu circa 80 de cuvinte cheie, și 12 tipuri de date predefinite. El permite programarea structurată, modulară și orientată obiectual, conform perceptelor moderne ale programării profesioniste.

Principiile de bază ale programării pe obiecte (INCAPSULARE, MOSTENIRE, POLIMORFISM) sunt elemente fundamentale ale programării C#. În mare, limbajul moștenește sintaxa și principiile de programare din C++. Sunt o serie de tipuri noi de date sau funcții diferite ale datelor din C++, iar în spiritul realizării unor secvențe de cod sigure (*safe*), unele funcții au fost adăugate (de exemplu, interfețe și delegări), diversificate (tipul *struct*), modificate (tipul *string*) sau chiar eliminate (moștenirea multiplă și pointerii către funcții). Unele funcții (cum ar fi accesul direct la memorie folosind pointeri) au fost păstrate, dar secvențele de cod corespunzătoare se consideră "nesigure".

3.2. Compilarea la linia de comandă

Se pot dezvolta aplicații .NET și fără a dispune de mediul de dezvoltare Visual Studio, ci numai de .NET SDK ([pentru 2.0](#) și [pentru 3.0](#)). În acest caz, codul se scrie în orice editor de text, fișierele se salvează cu extensia *cs*, apoi se compilează la linie de comandă.

Astfel, se scrie în Notepad programul:

```
using System;
class primul
{
    static void Main()
    {   Console.WriteLine("Primul program");
        Console.ReadKey(true);
    }
}
```

Dacă se salvează fișierul `primul.cs`, în directorul `WINDOWS\Microsoft.NET\Framework\v2.0`, atunci scriind la linia de comandă: `csc primul.cs` se va obține fișierul `primul.exe` direct executabil pe o platformă .NET.

3.3. Crearea aplicațiilor consolă

Pentru a realiza aplicații în mediul de dezvoltare Visual Studio, trebuie să instalăm o versiune a acestuia, eventual versiunea free **Microsoft Visual C# 2005 Express Edition** de la adresa <http://msdn.microsoft.com/vstudio/express/downloads/default.aspx>. Pentru început, putem realiza aplicații consolă (ca și cele din Borland Pascal sau Borland C).

După lansare, alegem opțiunea *New Project* din meniul *File*. În fereastra de dialog (vezi figura), selectăm pictograma **Console Application**, după care, la **Name**, introducem numele aplicației noastre.

Fereastra în care scriem programul se numește implicit **Program.cs** și se poate modifica prin salvare explicită (Save As). Extensia **cs** provine de la **C Sharp**.

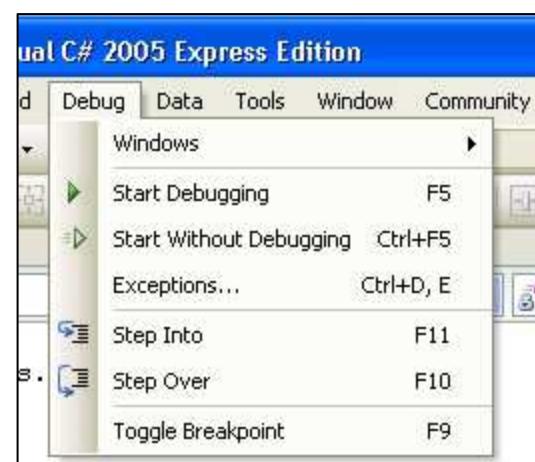
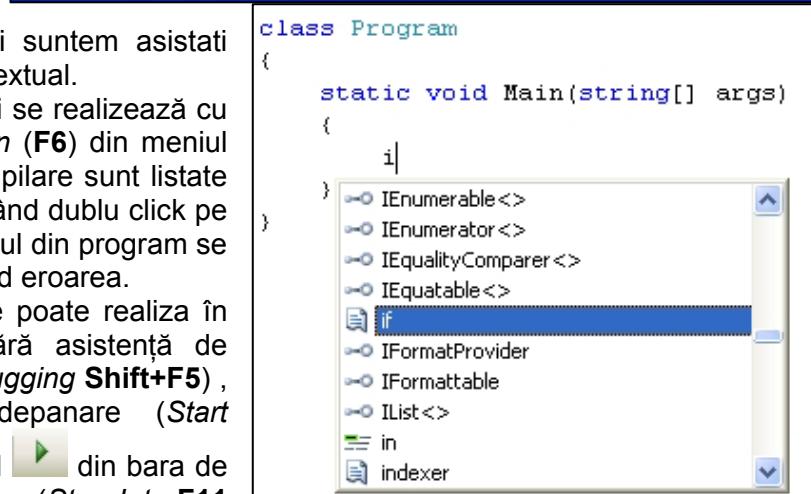
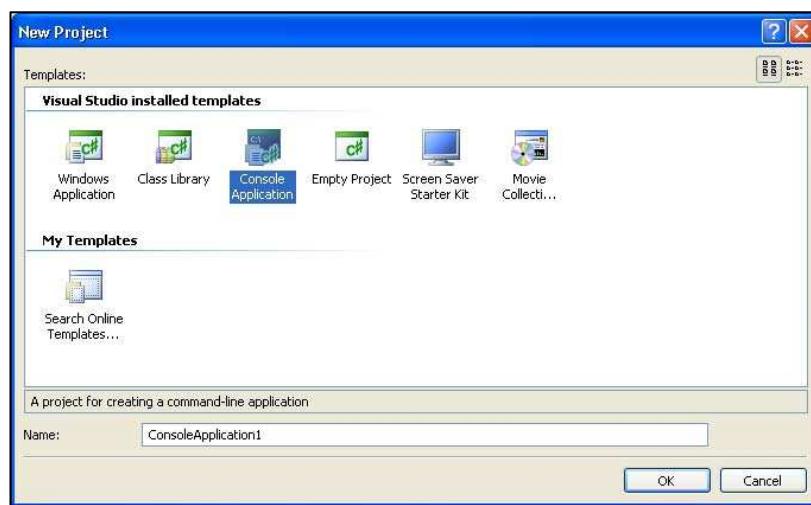
În scrierea programului suntem asistati de **IntelliSense**, ajutorul contextual.

Compilarea programului se realizează cu ajutorul opțiunii **Build Solution (F6)** din meniul *Build*. Posibilele erori de compilare sunt listate în fereastra *Error List*. Efectuând dublu click pe fiecare eroare în parte, cursorul din program se poziționează pe linia conținând eroarea.

Rularea programului se poate realiza în mai multe moduri: rapid fără asistență de depanare (*Start Without Debugging Shift+F5*), rapid cu asistență de depanare (*Start*

Debugging F5 sau cu butonul din bara de instrumente), rulare pas cu pas (*Step Into F11* și *Step Over F12*) sau rulare rapidă până la linia marcată ca punct de întrerupere (*Toggle Breakpoint F9* pe linia respectivă și apoi *Start Debugging F5*). Încetarea urmăririi pas cu pas (*Stop Debugging Shift+F5*) permite ieșirea din modul depanare și revenirea la modul normal de lucru. Toate opțiunile de rulare și depanare se găsesc în meniul *Debug* al meniului.

Fereastra de cod și ferestrele auxiliare ce ne ajută în etapa de editare pot fi vizualizate alegând opțiunea corespunzătoare din meniul *View*. Ferestrele auxiliare utile în etapa de depanare se pot vizualiza alegând opțiunea corespunzătoare din meniul *Debug/Windows*.



3.4. Structura unui program C#

Să începem cu exemplul clasic “Hello World” adaptat la limbajul C#:

```

1 using System;
2
3 namespace HelloWorld
4 {
5     class Program
6     {
7         static void Main()
8         {
9             Console.WriteLine("Hello World!");
10        }
11    }
12 }
```

O aplicație C# este formată din una sau mai multe **clase**, grupate în **spații de nume (namespaces)**. Un spațiu de nume cuprinde mai multe clase cu nume diferite având funcționalități înrudite. Două clase pot avea același nume cu condiția ca ele să fie definite în spații de nume diferite. În cadrul aceluiasi spațiu de nume poate apărea definiția unui alt spațiu de nume, caz în care avem de-a face cu spații de nume imbricate. O clasă poate fi identificată prin numele complet (nume precedat de numele spațiului sau spațiilor de nume din care face parte clasa respectivă, cu separatorul punct). În exemplul nostru, `HelloWorld.Program` este numele cu specificație completă al clasei `Program`.

O clasă este formată din date și metode (funcții). Apelarea unei metode în cadrul clasei în care a fost definită aceasta presupune specificarea numelui metodei. Apelul unei metode definite în interiorul unei clase poate fi invocată și din interiorul altrei clase, caz în care este necesară specificarea clasei și apoi a metodei separate prin punct. Dacă în plus, clasa aparține unui spațiu de nume neinclus în fișierul curent, atunci este necesară precizarea tuturor componentelor numelui: `spațiu.clasă.metodă` sau `spațiu.spațiu.clasă.metodă` etc.

În fișierul nostru se află două spații de nume: unul definit (`HelloWorld`) și unul extern inclus prin directiva `using (System)`. `Console.WriteLine` reprezintă apelul metodei `WriteLine` definită în clasa `Console`. Cum în spațiu de nume curent este definită doar clasa `Program`, deducem că definiția clasei `Console` trebuie să se găsească în spațiu `System`.

Pentru a facilita cooperarea mai multor programatori la realizarea unei aplicații complexe, există posibilitatea de a segmenta aplicația în mai multe fișiere numite **assemblies**. Într-un **assembly** se pot implementa mai multe spații de nume, iar parti ale unui același spațiu de nume se pot regăsi în mai multe assembly-uri. Pentru o aplicație consolă, ca și pentru o aplicație Windows de altfel, este obligatoriu ca una (și numai una) dintre clasele aplicației să conțină un „punct de intrare” (*entry point*), și anume metoda (funcția) `Main`.

Să comentăm programul de mai sus:

linia 1: este o directivă care specifică faptul că se vor folosi clase incluse în spațiu de nume `System`. În cazul nostru se va folosi clasa `Console`.

linia 3: spațiu nostru de nume

linia 5: orice program C# este alcătuit din una sau mai multe clase

linia 7: metoda `Main`, „punctul de intrare” în program

linia 9: clasa `Console`, amintită mai sus, este folosită pentru operațiile de intrare/ieșire. Aici se apelează metoda `WriteLine` din acestă clasă, pentru afișarea mesajului dorit pe ecran.

3.5. Sintaxa limbajului

Ca și limbajul C++ cu care se înrudește, limbajul C# are un alfabet format din litere mari și mici ale alfabetului englez, cifre și alte semne. Vocabularul limbajului este format din acele „simboluri”²⁷ cu semnificații lexicale în scrierea programelor: cuvinte (nume), expresii, separatori, delimitatori și comentarii.

²⁷ Este un termen folosit un pic echivoc și provenit din traduceriea cuvântului „token”

Comentarii

- **comentariu pe un rând** prin folosirea `//` Tot ce urmează după caracterele `//` sunt considerate, din acel loc, până la sfârșitul rândului drept comentariu
`// Aceasta este un comentariu pe un singur rand`
- **comentariu pe mai multe rânduri** prin folosirea `/* și */` Orice text cuprins între simbolurile menționate mai sus se consideră a fi comentariu. Simbolurile `/*` reprezintă începutul comentariului, iar `*/` sfârșitul respectivului comentariu.
`/* Aceasta este un
comentariu care se
intinde pe mai multe randuri */`

Nume

Prin **nume** dat unei variabile, clase, metode etc. înțelegem o succesiune de caractere care îndeplinește următoarele reguli:

- numele trebuie să înceapă cu o literă sau cu unul dintre caracterele `_` și `@`;
- primul caracter poate fi urmat numai de litere, cifre sau un caracter de subliniere;
- numele care reprezintă cuvinte cheie nu pot fi folosite în alt scop decât acela pentru care au fost definite
- cuvintele cheie pot fi folosite în alt scop numai dacă sunt precedate de `@`
- două nume sunt distințe dacă diferă prin cel puțin un caracter (fie el și literă mică ce diferă de aceeași literă majusculă)

Convenții pentru nume:

- în cazul numelor claselor, metodelor, a proprietăților, enumerărilor, interfețelor, spațiilor de nume, fiecare cuvânt care compune numele începe cu majusculă
- în cazul numelor variabilelor dacă numele este compus din mai multe cuvinte, primul începe cu minusculă, celelalte cu majusculă

Cuvinte cheie în C#

abstract	as	base	bool	break
byte	case	catch	char	checked
class	const	continue	decimal	default
delegate	do	double	else	enum
event	explicit	extern	false	finally
fixed	float	for	foreach	goto
if	implicit	in	int	interface
internal	is	lock	long	namespace
new	null	object	operator	out
override	params	private	protected	public
readonly	ref	return	sbyte	sealed
short	sizeof	stackalloc	static	string
struct	switch	this	throw	true
try	typeof	uint	ulong	unchecked
unsafe	ushort	using	virtual	void
volatile	while			

Simbolurile lexicale reprezentând constante, regulile de formare a expresiilor, separatorii de liste, delimitatorii de instrucțiuni, de blocuri de instrucțiuni, de siruri de caractere etc. sunt în mare aceiași ca și în cazul limbajului C++.

3.6. Tipuri de date

În C# există două categorii de tipuri de date:

- **tipuri valoare**
 - tipul simple: **byte**, **char**, **int**, **float** etc.
 - tipul enumerare - **enum**
 - tipul structură - **struct**

- **tipuri referință**
 - tipul clasă - **class**
 - tipul interfață - **interface**
 - tipul delegat - **delegate**
 - tipul tablou - **array**

Toate tipurile de date sunt derive din tipul **System.Object**

Toate tipurile **valoare** sunt derive din clasa **System.ValueType**, derivată la rândul ei din clasa **Object** (alias pentru **System.Object**).

Limbajul C# conține un set de **tipuri predefinite** (int, bool etc.) și permite definirea unor tipuri proprii (enum, struct, class etc.).

Tipuri simple predefinite

Tip	Descriere	Domeniul de valori
object	rădăcina oricărui tip	
string	secvență de caractere Unicode	
sbyte	tip întreg cu semn, pe 8 biți	-128; 127
short	tip întreg cu semn, pe 16 biți	-32768; 32767
int	tip întreg cu semn pe, 32 biți	-2147483648; 21447483647
long	tip întreg cu semn, pe 64 de biți	-9223372036854775808; 9223372036854775807
byte	tip întreg fără semn, pe 8 biți	0; 255
ushort	tip întreg fără semn, pe 16 biți	0; 65535
uint	tip întreg fără semn, pe 32 biți	0; 4294967295
ulong	tip întreg fără semn, pe 64 biți	0; 18446744073709551615
float	tip cu virgulă mobilă, simplă precizie, pe 32 biți (8 pentru exponent, 24 pentru mantisă)	-3.402823E+38; 3.402823E+38
double	tip cu virgulă mobilă, dublă precizie, pe 64 biți (11 pentru exponent, 53 -mantisa)	-1.79769313486232E+308; 1.79769313486232E+308
bool	tip boolean	- 79228162514264337593543950335; 79228162514264337593543950335
char	tip caracter din setul Unicode, pe 16 biți	
decimal	tip zecimal, pe 128 biți (96 pentru mantisă), 28 de cifre semnificative	

O valoare se asignează după următoarele reguli:

Sufix	Tip
nu are	int, uint, long, ulong
u, U	uint, ulong
L, L	long, ulong
ul, lu, UL, LU, UL, LU, Lu	ulong

Exemple:

<pre>string s = "Salut!"</pre>	<pre>float g = 1.234F;</pre>
<pre>long a = 10;</pre>	<pre>double h = 1.234;</pre>
<pre>long b = 13L;</pre>	<pre>double i = 1.234D;</pre>
<pre>ulong c = 12;</pre>	<pre>bool cond1 = true;</pre>
<pre>ulong d = 15U;</pre>	<pre>bool cond2 = false;</pre>
<pre>ulong e = 16L;</pre>	<pre>decimal j = 1.234M;</pre>
<pre>ulong f = 17UL;</pre>	

Tipul enumerare

Tipul enumerare este un tip de finit de utilizator. Acest tip permite utilizarea numelor care, sunt asociate unor valori numerice. Toate componentele enumerate au un același tip de bază întreg. În cazul în care, la declarare, nu se specifică tipul de bază al enumerării, atunci acesta este considerat implicit **int**.

Declararea unui tip enumerare este de forma:

```
enum [Nume_tip] [: Tip].
{
    [identificator1][=valoare].,
    ..
    [identificatorn][=valoare].
}
```

Observații:

- În mod implicit valoarea primului membru al enumerării este 0, iar fiecare variabilă care urmează are valoarea (implicită) mai mare cu o unitate decât precedenta.
- Valorile folosite pentru inițializări trebuie să facă parte din domeniul de valori declarat al tipului
- Nu se admit referințe circulare:

```
enum ValoriCirculare
{
    a = b,
    b
}
```

Exemplu:

```
using System;
namespace tipulEnum
{
    class Program
    {
        enum lunaAnului
        {
            Ianuarie = 1,
            Februarie, Martie, Aprilie, Mai, Iunie, Iulie,
            August, Septembrie, Octombrie, Noiembrie, Decembrie
        }
        static void Main(string[] args)
        {
            Console.WriteLine("Luna Mai este a ",
                (int)lunaAnului.Mai + "a luna din an.");
            Console.ReadLine();
        }
    }
}
```

În urma rulării programului se afișează mesajul :

Luna Mai este a 5 a luna din an.

Tablouri

Declararea unui tablou unidimensional:

```
Tip[] nume;
```

Prin aceasta, **nu** se alocă spațiu pentru memorare. Pentru a putea reține date în structura de tip tablou, este necesară o operație de **instantiere**:

```
nume = new Tip[NumarElemente];
```

Declararea, instantierea și chiar inițializarea tabloului se pot face în aceeași instrucțiune:

Exemplu:

```
int[] v = new int[] {1,2,3}; sau
int[] v = {1,2,3}; //new este implicit
```

În cazul tablourilor cu mai multe dimensiuni facem distincție între **tablouri regulate** și **tablouri neregulate (tablouri de tablouri)**

Declarare în cazul **tablourilor regulate bidimensionale**:

Tip[,] nume;

Întâișere:

```
nume = new Tip[Linii,Coloane];
```

Acces:

```
nume[indice1,indice2]
```

Exemplu:

```
int[,] mat = new int[,] {{1,2,3},{4,5,6},{7,8,9}}; sau  
int[,] mat = {{1,2,3},{4,5,6},{7,8,9}};
```

Declarare în cazul **tablourilor neregulate bidimensionale**:

Tip[][] nume;

Întâișere:

```
nume = new Tip[Linii][];
```

```
nume[0]=new Tip[Coloane]
```

...

```
nume[Linii-1]=new Tip[ColoaneLinii-1]
```

Acces:

```
nume[indice1][indice2]
```

Exemplu:

```
int[][][] mat = new int[][][] {  
    new int[3] {1,2,3},  
    new int[2] {4,5},  
    new int[4] {7,8,9,1}  
}; sau  
int[][][] mat={{new int[3] {1,2,3},new int[2] {4,5},new int[4] {7,8,9,1}}};
```

Şiruri de caractere

Se definesc două tipuri de şiruri:

- regulate
- de tip „**verbatim**”

Tipul regulat conține între ghilimele zero sau mai multe caractere, inclusiv secvențe escape.

Secvențele escape permit reprezentarea caracterelor care nu au reprezentare grafică precum și reprezentarea unor caractere speciale: backslash, caracterul apostrof, etc.

Secvență escape	Efect
\'	apostrof
\”	ghilimele
\\\	backslash
\0	null
\a	alarmă
\b	backspace
\f	form feed – pagină nouă
\n	new line – linie nouă
\r	carriage return – început de rând
\t	horizontal tab – tab orizontal
\u	caracter unicode
\v	vertical tab – tab vertical
\x	caracter hexazecimal

În cazul în care folosim multe secvențe escape, putem utiliza şirurile **verbatim**. Aceste şiruri pot să conțină orice fel de caractere, inclusiv caracterul EOLN. Ele se folosesc în special în cazul în care dorim să facem referiri la fișiere și la regiștri. Un astfel de șir începe întotdeauna cu simbolul '@' înaintea ghilimelelor de început.

Exemplu:

```
using System;
namespace SiruriDeCaractere
{
    class Program
    {
        static void Main(string[] args)
        {
            string a = "un sir de caractere";
            string b = "linia unu \nlinia doi";
            string c = @"linia unu
linia doi";
            string d="c:\\exemple\\unu.cs";
            string e = @"c:\\exemple\\unu.cs";
            Console.WriteLine(a); Console.WriteLine(b);
            Console.WriteLine(c); Console.WriteLine(d);
            Console.WriteLine(e); Console.ReadLine();
        }
    }
}
```

Programul va avea ieșirea

```
un sir de caractere
linia unu
linia doi
linia unu
linia doi
c:\\exemple\\unu.cs
c:\\exemple\\unu.cs
```

3.7. Conversii

Conversii numerice

În C# există două tipuri de conversii numerice:

- **implicite**
- **explicite.**

Conversia implicită se efectuează (automat) doar dacă nu este afectată valoarea convertită. Regulile de **conversie implicită** sunt descrise de tabelul următor:

din	în
sbyte	short, int, long, float, double, decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
char	ushort, int, uint, long, ulong, float, double, decimal
float	double
ulong	float, double, decimal

Conversia explicită se realizează prin intermediul unei expresii cast, atunci când nu există posibilitatea unei conversii隐式.

Exemplu:

```
int i=Console.Read(); char c;
c=char(i);
```

Conversii boxing și unboxing

Datorită faptului că în C# toate tipurile sunt derive din clasa **Object** (**System.Object**), prin conversiile **boxing** (împachetare) și **unboxing** (despachetare) este permisă tratarea tipurilor valoare drept obiecte și reciproc. Prin conversia boxing a unui tip valoare, care se păstrează pe stivă, se produce ambalarea în interiorul unei instanțe de tip referință, care se păstrază în memoria heap, la clasa **Object**. Unboxing permite convertirea unui obiect într-un tip valoare corespunzător.

Exemplu:

Prin boxing variabila **i** este asignata unui obiect **ob**:

```
int i = 13;
object ob = (object)i; //boxing explicit
sau
```

```
int i = 13;
object ob = i; //boxing implicit
```

Prin conversia de tip unboxing, obiectul **ob** poate fi asignat variabilei întregi **i**:

```
int i=13;
object ob = i; //boxing implicit
i = (int)ob; //unboxing explicit
```

3.8. Constante

În C# există două modalități de declarare a constantelor: folosind **const** sau folosind modificatorul **readonly**. Constantele declarate cu **const** trebuie să fie inițializate la declararea lor.

Exemplu:

```
const int x; //gresit, constanta nu a fost initializata
const int x = 13; //corect
```

3.9. Variabile

O variabilă în C# poate să conțină fie o valoare a unui tip elementar, fie o referință la un obiect.

Exemplu:

```
int Salut;
int Azi_si _maine;
char caracter;
```

3.10. Expresii și operatori

Prin **expresie** se înțelege o secvență formată din **operatori** și **operanzi**. Un **operator** este un simbol ce indică acțiunea care se efectuează, iar **operandul** este valoarea asupra căreia se execută operația.

În C# sunt definiți mai mulți operatori. În cazul în care într-o expresie nu intervin paranteze, operațiile se execută conform priorității operatorilor. În cazul în care sunt mai mulți operatori cu aceeași prioritate, evaluarea expresiei se realizează de la stânga la dreapta.

Prioritate	Tip	Operatori	Asociativitate
0	Primar	() [] f() . x++ x-- new typeof sizeof checked unchecked ->	→
1	Unar	+ - ! ~ ++x --x (tip) true false & sizeof	→
2	Multiplicativ	* / %	→
3	Aditiv	+ -	→
4	De deplasare	<< >>	→
5	Relațional	< > <= >= is as	→
6	De egalitate	== !=	→

7	AND (SI) logic	&	→
8	XOR (SAU exclusiv) logic	^	→
9	OR (SAU) logic		→
10	AND (SI) conditional	&&	→
11	OR (SAU) conditional		→
12	Conditional	?:	←
13	De atribuire	= *= /= %= += -= ^= &= <<= >>= =	←

3.11. Colecții

O **colecție** în C# este o clasă specializată pentru memorarea și regăsirea rapidă a informațiilor. Ea implementează metode specifice de actualizare dinamică colecției, de căutare și de enumerare a datelor. Cel mai des folosite sunt colecțiile de obiecte și colecțiile generice (vezi capitolul 5.1) cum ar fi liste, stive, hash-uri. Aceste clase pot fi derivate pentru a obține colecții specializate care se potrivesc cel mai bine necesităților de memorare a datelor specifice unei aplicații. Colecțiile sunt definite în spațiul System.Collection. Metodele uzuale ale claselor din spațiul Collection sunt: *Add*, *Remove*, *IndexOf*, *Sort*, *Reverse*, *CopyToArray*, *Find*, *Foreach* etc.

3.12. Instrucțiunea foreach

Cum instrucțiunile de apel, atribuire, decizie, selecție și trei structuri repetitive coincid ca formă și funcționalitate cu cele din C, ne oprim sumar numai unora dintre noile structuri de control specifice limbajului C#.

Instrucțiunea foreach enumeră elementele dintr-o colecție sau dintr-un tablou, executând un bloc de instrucțuni pentru fiecare element al colecției sau tabloului. La fiecare iterare, elementul curent al colecției este de tip *readonly*, neputând fi modificat. Amintim că în instrucțiunea repetitivă **foreach** se pot utiliza cu exact aceeași funcționalitate instrucțiunile de salt **break** și **continue**.

instrucțiunea se utilizează curent în aplicații pentru tablouri și colecții de obiecte²⁸.

Pentru a vedea cum acționează o vom compara cu instrucțiunea cunoscută **for**. Fie vectorul nume format din siruri de caractere:

```
string[] nume = { "Ana", "Ionel", "Maria" };
```

Să afișăm acest sir folosind instrucțiunea **for**:

```
for(int i=0; i<nume.Length; i++)
{
    Console.WriteLine("{0} ", nume[i]);
}
```

Același rezultat îl obținem folosind instrucțiunea **foreach**:

```
foreach (string copil in nume)
{
    Console.WriteLine("{0} ", copil);
}
```

3.13. Instrucțiunile try-catch-finally și throw

Prin **excepție** se înțelege un obiect care încapsulează informații despre situații anormale în funcționarea unui program. Ea se folosește pentru a semnala contextul în care

²⁸ de exemplu, pentru prelucrarea sistematică a tuturor componentelor unei ferestre (butoane, liste, casete de text etc.)

apare o situație specială. De exemplu: erori la deschiderea unor fișiere, împărțire la 0 etc. Aceste erori se pot manipula astfel încât programul să nu se termine abrupt.

Sunt situații în care prefigurăm apariția unei erori într-o secvență de prelucrare și atunci integrăm secvența respectivă în blocul unei instrucțiuni **try**, precizând una sau mai multe secvențe de program pentru tratarea excepțiilor apărute (blocuri **catch**) și eventual o secvență comună care se execută după terminarea normală sau după "recuperarea" programului din starea de excepție (blocul **finally**).

Exemplu:

```
using System;
using System.IO;
class tryCatch
{
    static void Main(string[] args)
    {
        string s;
        Console.WriteLine("Numele fisierului:");
        Console.ReadLine(s);
        try
        {
            File.OpenRead(s);
        }
        catch (FileNotFoundException a)
        {
            Console.WriteLine(a.ToString());
        }
        catch (PathTooLongException b)
        {
            Console.WriteLine(b.ToString());
        }
        finally
        {
            Console.WriteLine("Programul s-a sfarsit");
            Console.ReadLine();
        }
    }
}
```

Alteori putem simula prin program o stare de eroare "aruncând" o excepție (instrucțiunea **throw**) sau putem profita de mecanismul de tratare a erorilor pentru a implementa un sistem de validare a datelor prin generarea unei excepții proprii pe care, de asemenea, o "aruncăm" în momentul neîndeplinirii unor condiții puse asupra datelor.

Clasa **System.Exception** și derivate ale acesteia servesc la tratarea adecvată și diversificată a excepțiilor.

Exemplu: Considerăm clasele Copil, Fetita, Baiat definite fragmentat în capitolul 1. O posibilitate de validare la adăugarea unui copil este aceea care generează o excepție proprie la depășirea dimensiunii vectorului static **copii**:

```
public static void adaug_copil(Copil c)
{
    if (nr_copii < nr_max)
        copii[nr_copii++] = c;
    else throw new Exception("Prea multi copii");
}
```

4. Programare vizuală

4.1. Concepte de bază ale programării vizuale

Programarea vizuală trebuie privită ca un mod de proiectare a unui program prin operare directă asupra unui set de elemente grafice (de aici vine denumirea de programare vizuală). Această operare are ca efect scrierea automată a unor secvențe de program, secvențe care, împreună cu secvențele scrise textual²⁹, vor forma programul.

Spunem că o *aplicație* este *vizuală* dacă dispune de o interfață grafică sugestivă și pune la dispoziția utilizatorului instrumente specifice de utilizare (*drag*, *click*, *hint* etc.)

Realizarea unei aplicații vizuale nu constă doar în desenare și aranjare de controale, ci presupune în principal stabilirea unor decizii arhitecturale³⁰, decizii ce au la bază unul dintre **modelele arhitecturale** de bază:

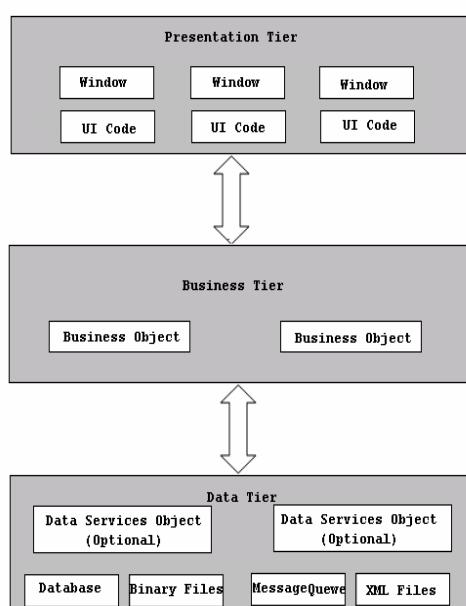
a) Modelul arhitectural **orientat pe date**.

Acest model nu este orientat pe obiecte, timpul de dezvoltare al unei astfel de aplicații este foarte mic, o parte a codului este generată automat de Visual Studio.NET, codul nu este foarte ușor de întreținut și este recomandat pentru aplicații relativ mici sau cu multe operații de acces (in/out) la o bază de date.

b) Modelul arhitectural **Model-view-controller**

Este caracterizat de cele trei concepte de bază : **Model** (reprezentarea datelor se realizează într-o manieră specifică aplicației: conține obiectele de „business”, încapsulează accesul la date), **View** (sunt utilizate elemente de interfață, este format din Form-uri), **Controller**(procesează și răspunde la evenimente iar SO, clasele Form și Control din .Net rutează evenimentul către un „handler”, eveniment tratat în codul din spatele Form-urilor).

c) Modelul arhitectural **Multi-nivel**



Nivelul de prezentare (interfață)

Se ocupă numai de afișarea informațiilor către utilizator și captarea celor introduse de acesta. Nu cuprinde detalii despre logica aplicației, și cu atât mai mult despre baza de date sau fișierele pe care aceasta le utilizează. Cu alte cuvinte, în cadrul interfeței cu utilizatorul, nu se vor folosi obiecte de tipuri definite de programator, ci numai baza din .NET.

Nivelul de logică a aplicației

Se ocupă de tot ceea ce este specific aplicației care se dezvoltă. Aici se efectuează calculele și procesările și se lucrează cu obiecte de tipuri definite de programator.

Nivelul de acces la date

Aici rezidă codul care se ocupă cu accesul la baza de date, la fișiere, la alte servicii.

Această ultimă structură este foarte bună pentru a organiza aplicațiile, dar nu este ușor de realizat. De exemplu, dacă în interfață cu utilizatorul prezentăm date sub formă *ListView* și la un moment dat clientul ne cere reprezentarea datelor într-un *GridView*, modificările la nivel de cod nu se pot localiza doar în interfață deoarece cele două controale au nevoie de modele de acces la date total diferite.

²⁹ Se utilizează adesea antonimia dintre vizual (operații asupra unor componente grafice) și textual (scriere de linii de cod); proiectarea oricărei aplicații "vizuale" îmbină ambele tehnici.

³⁰ Deciziile arhitecturale stabilesc în principal cum se leagă interfața de restul aplicației și cât de ușor de întreținut este codul rezultat.

Indiferent de modelul arhitectural ales, în realizarea aplicației mai trebuie respectate și **principiile proiectării interfețelor**:

- **Simplitatea**

Interfața trebuie să fie cât mai ușor de înțeles³¹ și de învățat de către utilizator și să permită acestuia să efectueze operațiile dorite în timp cât mai scurt. În acest sens, este vitală culegerea de informații despre utilizatorii finali ai aplicației și a modului în care aceștia sunt obișnuiți să lucreze.

- **Pozitia controalelor**

Locația controalelor dintr-o fereastră trebuie să reflecte importanța relativă și frecvența de utilizare. Astfel, când un utilizator trebuie să introducă niște informații – unele obligatorii și altele opționale – este indicat să organizăm controalele astfel încât primele să fie cele care preiau informații obligatorii.

- **Consistența**

Ferestrele și controalele trebuie să fie afișate după un design asemănător („template”) pe parcursul utilizării aplicației. Înainte de a implementa interfața, trebuie decidem cum va arăta aceasta, să definim „template”-ul.

- **Estetica**

Interfața trebuie să fie pe cât posibil plăcută și atrăgătoare.

4.2. Mediul de dezvoltare Visual C#

Mediul de dezvoltare **Microsoft Visual C#** dispune de instrumente specializate de proiectare, ceea ce permite crearea aplicațiilor în mod interactiv, rapid și ușor.

Pentru a construi o aplicație Windows (File→New Project) se selectează ca template *Windows Application*.

O aplicație Windows conține cel puțin o fereastră (*Form*) în care se poate crea o interfață cu utilizatorul aplicației.

Componentele vizuale ale aplicației pot fi prelucrate în modul **Designer** (**Shift+F7**) pentru a plasa noi obiecte, a le stabili proprietățile etc. Codul "din spatele" unei componente vizuale este accesibil în modul **Code** (**F7**).

În fereastra **Solution Explorer** sunt afișate toate fișierele pe care Visual Studio.NET le-a inclus în proiect. **Form1.cs** este formularul creat implicit de Visual Studio.NET ca parte a proiectului.

Fereastra **Properties** este utilizată pentru a vizualiza și eventual schimba proprietățile obiectelor.

Toolbox conține **controale standard drag-and-drop** și componente utilizate în crearea aplicației Windows. Controalele sunt grupate în categoriile logice din imaginea alăturată.



Designer, **Code**, **Solution Explorer** și celelalte se află grupate în meniu **View**.

La crearea unei noi aplicații vizuale, Visual Studio.NET generează un spațiu de nume ce conține clasa statică *Program*, cu metoda statică ce constituie punctul de intrare (de lansare) a aplicației:

```
static void Main()
{
    ...
    Application.Run(new Form1());
}
```

Clasa Application este responsabilă cu administrarea unei aplicații Windows, punând la dispoziție proprietăți pentru a obține informații despre aplicație, metode de lucru cu aplicația și altele. Toate metodele și proprietățile clasei Application sunt **static**. Metoda *Run* invocată

³¹ Întrucât mintea umană poate să perceapă la un moment dat aproximativ 5-9 obiecte, o fereastră supra-încărcată de controale o face greu de utilizat.

mai sus creează un formular implicit, aplicația răspunzând la mesajele utilizatorului până când formularul va fi închis.

Compilarea modulelor aplicației și asamblarea lor într-un singur fișier "executabil" se realizează cu ajutorul opțiunilor din meniu Build, uzuală fiind **Build Solution (F6)**.

Odată implementată, aplicația poate fi lansată, cu asistență de depanare sau nu (opțiunile **Start** din meniu *Debug*). Alte facilități de depanare pot fi folosite prin umărirea pas cu pas, umărirea până la puncte de intrerupere etc. (celelalte opțiuni ale meniului *Debug*). Ferestre auxiliare de umărire sunt vizualizate automat în timpul procesului de depanare, sau pot fi activate din submeniul *Windows* al meniului *Debug*.

4.3. Ferestre

Spațiul Forms ne oferă clase specialize pentru: creare de fereestre sau *formulare* (**System.Windows.Forms.Form**), elemente specifice (*controale*) cum ar fi butoane (**System.Windows.Forms.Button**), casete de text (**System.Windows.Forms.TextBox**) etc. Proiectarea unei fereestre are la bază un cod complex, generat automat pe măsură ce noi desemnăm componentele și comportamentul acestelor. În fapt, acest cod realizează: derivarea unei clase proprii din **System.Windows.Forms.Form**, clasă care este înzestrată cu o *colecție de controale* (initial vidă). Constructorul ferestrăi realizează instanțieri ale claselor *Button*, *MenuStrip*, *Timer* etc. (orice plasăm noi în fereastră) și adaugă referințele acestor obiecte la colecția de controale ale ferestrăi.

Dacă modelul de fereastră reprezintă fereastra principală a aplicației, atunci ea este instanțiată automat în programul principal (metoda *Main*). Dacă nu, trebuie să scriem noi codul ce realizează instanțierea.

Clasele derivate din *Form* moștenesc o serie de proprietăți care determină atributele vizuale ale ferestrăi (stilul marginilor, culoare de fundal, etc.), metode care implementează anumite comportamente (*Show*, *Hide*, *Focus* etc.) și o serie de metode specifice (*handle*re) de tratare a evenimentelor (*Load*, *Click* etc.).

O fereastră poate fi activată cu **form.Show()** sau cu **form.ShowDialog()**, metoda a doua permitând ca revenirea în fereastra din care a fost activat noul formular să se facă numai după ce noul formular a fost inchis (spunem că formularul nou este deschis **modal**).

Un **proprietar** este o fereastră care contribuie la comportarea formularului detinut. Activarea proprietarului unui formular deschis modal va determina activarea formularului deschis modal. Când un nou formular este activat folosind **form.Show()** nu va avea nici un detinător, acesta stabilindu-se direct :

```
public Form Owner { get; set; }
F_nou form=new F_nou();
form.Owner = this; form.Show();
```

Formularul deschis modal va avea un proprietar setat pe **null**. Detinătorul se poate stabili setând proprietarul înainte să apelăm **Form.ShowDialog()** sau apelând **From.ShowDialog()** cu proprietarul ca argument.

```
F_nou form = new F_nou();form.ShowDialog(this);
```

Vizibilitatea unui formular poate fi setată folosind metodele **Hide** sau **Show**. Pentru a ascunde un formular putem folosi :

```
this.Hide(); // setarea proprietății Visible indirect sau
this.Visible = false; // setarea proprietății Visible direct
```

Printre cele mai uzuale proprietăți ale form-urilor, reamintim:

- **StartPosition** determină poziția ferestrăi atunci când aceasta apare prima dată, poziție ce poate fi setată **Manual** sau poate fi centrată pe desktop (**CenterScreen**), stabilită de Windows, formularul având dimensiunile și locația stabilite de programator (**WindowsDefaultLocation**) sau Windows-ul va stabili dimensiunea inițială și locația pentru formular (**WindowsDefaultBounds**) sau, centrat pe formularul care l-a afișat (**CenterParent**) atunci când formularul va fi afișat modal.

- **Location (X,Y)** reprezintă coordonatele colțului din stânga sus al formularului relativ la colțul stânga sus al containerului. (Această proprietate e ignorată dacă StartPosition = Manual).

Mișcarea formularului (și implicit schimbarea locației) poate fi tratată în evenimentele **Move** și **LocationChanged**.

Locația formularului poate fi stabilită relativ la desktop astfel:

```
void Form_Load(object sender, EventArgs e) {
    this.Location = new Point(1, 1);
    this.DesktopLocation = new Point(1, 1); } //formularul in desktop
```

- **Size (Width și Height)** reprezintă dimensiunea ferestrei. Când se schimbă proprietățile Width și Height ale unui formular, acesta se va redimensiona automat, această redimensionare fiind tratată în evenimentele **Resize** sau în **SizeChanged**.

Chiar dacă proprietatea **Size** a formularului indică dimensiunea ferestrei, formularul nu este în totalitate responsabil pentru desenarea întregului conținut al său. Partea care este desenată de formular mai este denumită și Client Area. Marginile, titlul și scrollbar-ul sunt desenate de Windows.

- **MaximizeBox** și **MinimizeBox** sunt utilizate pentru a restricționa dimensiunile unui formular.

```
void Form_Load(object sender, EventArgs e) {
    this.MinimumSize = new Size(200, 100);
    this.MaximumSize = new Size(int.MaxValue, 100); }
```

- **IsMdiContainer** precizează dacă form-ul reprezintă un container pentru alte form-uri.
- **ControlBox** precizează dacă fereastra conține sau nu un icon, butonul de închidere al ferestrei și meniul System (Restore, Move, Size, Maximize, Minimize, Close).
- **HelpButton**-precizează dacă butonul  va apărea sau nu lângă butonul de închidere al formularului (doar dacă MaximizeBox=false, MinimizeBox=false). Dacă utilizatorul apasă acest buton și apoi apasă oriunde pe formular va apărea evenimentul **HelpRequested** (F1).
- **Icon** reprezintă un obiect de tip *.ico folosit ca icon pentru formular.
- **MaximizeBox** și **MinimizeBox** precizează dacă fereastra are sau nu butonul Maximize și respectiv Minimize
- **Opacity** indică procentul de opacitate³²
- **ShowInTaskbar** precizează dacă fereastra apare în TaskBar atunci când formularul este minimizat.
- **SizeGripStyle** specifică tipul pentru 'Size Grip' (Auto, Show, Hide). Size grip  (în colțul din dreapta jos) indică faptul că această fereastră poate fi redimensionată.
- **TopMost** precizează dacă fereastra este afisată în fața tuturor celorlalte ferestre.
- **TransparencyKey** identifică o culoare care va deveni transparentă pe formă.

Definirea unei funcții de tratare a unui **eveniment** asociat controlului se realizează prin selectarea grupului  *Events* din ferestra *Properties* a controlului respectiv și alegerea evenimentului dorit.

Dacă nu scriem nici un nume pentru funcția de tratare, ci efectuăm dublu click în căsuța respectivă, se generează automat un nume pentru această funcție, ținând cont de numele controlului și de numele evenimentului (de exemplu `button1_Click`).

Dacă în **Designer** efectuăm dublu click pe un control, se va genera automat o funcție de tratare pentru evenimentul implicit asociat controlului (pentru un buton evenimentul implicit este *Click*, pentru *TextBox* este *TextChanged*, pentru un formular *Load* etc.).

Printre **evenimentele** cele mai des utilizate, se numără :

- **Load** apare când formularul este pentru prima dată încărcat în memorie.

³² Dacă va fi setată la 10% formularul și toate controalele sale vor fi aproape invizibile.

- **FormClosed** apare când formularul este închis.
- **FormClosing** apare când formularul se va închide ca rezultat al acțiunii utilizatorului asupra butonului Close (Dacă se setează CancelEventArgs.Cancel =True atunci se va opri închiderea formularului).
- **Activated** apare pentru formularul activ.
- **Deactivate** apare atunci când utilizatorul va da click pe alt formular al aplicației.

4.4. Controale

Unitatea de bază a unei interfețe Windows o reprezintă un control. Acesta poate fi „găzduit” de un container ce poate fi un formular sau un alt control.

Un control este o instanță a unei clase derivate din System.Windows.Forms și este responsabil cu desenarea unei părți din container. Visual Studio .NET vine cu o serie de controale standard, disponibile în Toolbox. Aceste controale pot fi grupate astfel:

- **Controale de acțiune** (de exemplu **button**) care, atunci când sunt acționate, se poate executa o prelucrare. De exemplu, cel mai important eveniment pentru **Button** este **Click** (desemnând acțiunea click stânga pe buton).

În exemplul **PV1** se adaugă pe formular două butoane și o casetă text. Apăsarea primului buton va determina afișarea textului din TextBox într-un MessageBox iar apăsarea celui de-al doilea buton va închide încă aplicația. După adăugarea celor două butoane și a casetei text a fost schimbat textul afișat pe cele două butoane au fost scrise funcțiile de tratare a evenimentului **Click** pentru cele două butoane:

```
private void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show(textBox1.Text);
}
private void button2_Click(object sender, System.EventArgs e)
{
    Form1.ActiveForm.Dispose();
}
```



- **Controale valoare** (**label**, **textbox**, **picturebox**) care arată utilizatorului o informație (text, imagine).

Label este folosit pentru plasarea de text pe un formular. Textul afișat este conținut în proprietatea **Text** și este aliniat conform proprietății **TextAlign**.

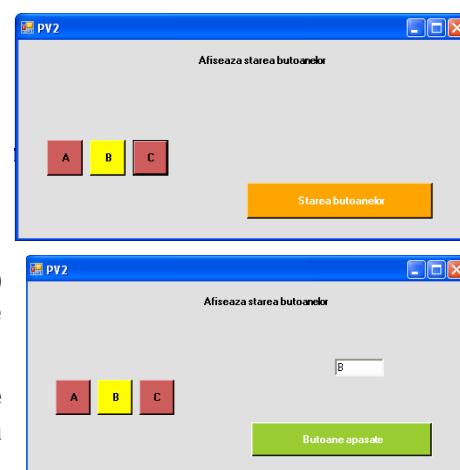
TextBox - permite utilizatorului să introducă un text. Prevede, prin intermediul ContextMenu-ului asociat, un set de funcționalități de bază, ca de exemplu (Cut, Copy, Paste, Delete, SelectAll).

PictureBox permite afișarea unei imagini.

Exemplul PV2 afișează un grup alcătuit din 3 butoane, etichetate A,B respectiv C având inițial culoarea roșie. Apăsarea unui buton determină schimbarea culorii acestuia în galben. La o nouă apăsare butonul revine la culoare inițială. Acționarea butonului “Starea butoanelor” determină afișarea într-o casetă text a etichetelor butoanelor galbene. Caseta text devine vizibilă atunci când apăsăm prima oară acest buton. Culoarea butonului mare (verde/portocaliu) se schimbă atunci când mouse-ul este poziționat pe buton.

După adăugarea butoanelor și a casetei text pe formular, stabilim evenimentele care determină schimbarea culorilor și completarea casetei text.

```
private void button1_Click(object sender, System.EventArgs e)
{
    if (button1.BackColor== Color.IndianRed) button1.BackColor=Color.Yellow;
    else button1.BackColor= Color.IndianRed;
}
```



```

private void button4_MouseEnter(object sender, System.EventArgs e)
    {button4.BackColor=Color.YellowGreen;button4.Text="Butoane apasate";}
private void button4_MouseLeave(object sender, System.EventArgs e)
    {textBox1.Visible=false;button4.Text="Starea butoanelor";
     button4.BackColor=Color.Orange;}
private void button4_Click(object sender, System.EventArgs e)
    {textBox1.Visible=true;textBox1.Text="";
     if( button1.BackColor==Color.Yellow)textBox1.Text=textBox1.Text+'A';
     if( button2.BackColor==Color.Yellow)textBox1.Text=textBox1.Text+'B';
     if( button3.BackColor==Color.Yellow)textBox1.Text=textBox1.Text+'C';
    }

```

Exercitiu Modificați aplicația precedentă astfel încât să avem un singur eveniment button_Click, diferențierea fiind făcută de parametrul sender.

Exercitău (Password) Adăugați pe un formular o casetă text în care să introduceți un sir de caractere și apoi verificați dacă acesta coincide cu o parolă dată. Textul introdus în casetă nu este vizibil (fiecare caracter este înlocuit cu *). Rezultatul va fi afișat într-un MessageBox.

- **Controale de selecție (CheckBox,RadioButton)** au proprietatea **Checked** care indică dacă am selectat controlul. După schimbarea stării unui astfel de control, se declanșează evenimentul **Checked**. Dacă proprietatea **ThreeState** este setată, atunci se schimbă funcționalitatea acestor controale, în sensul că acestea vor permite setarea unei alte stări. În acest caz, trebuie verificată proprietatea **CheckState(Checked, Unchecked,Indeterminate)** pentru a vedea starea controlului.

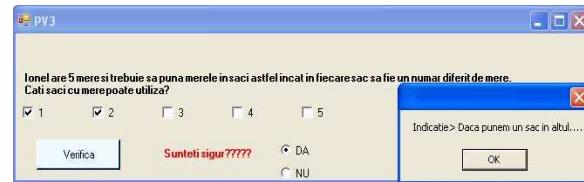
Aplicația **PV3** este un exemplu de utilizare a acestor controale. Soluția unei probleme cu mai multe variante de răspuns este memorată cu ajutorul unor checkbox-uri cu proprietatea **ThreeState**. Apăsarea butonului **Verifică** determină afișarea unei etichete și a butoanelor radio DA și NU. Răspunsul este afișat într-un MessageBox.

După adăugarea controalelor pe formular și setarea proprietăților **Text** și **ThreeState** în cazul checkbox-urilor stabilim evenimentele click pentru butonul Verifica și pentru butonul radio cu eticheta DA:

```

private void radioButton1_Click(object sender, System.EventArgs e)
{if (checkBox1.CheckState==CheckState.Checked &&
    checkBox2.CheckState==CheckState.Checked &&
    checkBox3.CheckState==CheckState.Checked &&
    checkBox5.CheckState==CheckState.Checked &&
    checkBox4.CheckState==CheckState.Unchecked) MessageBox.Show("CORECT");
else MessageBox.Show("Indicatie> Daca punem un sac in altul....");
label2.Visible=false;
radioButton1.Checked=false; radioButton2.Checked=false;
radioButton1.Visible=false; radioButton2.Visible=false;}
private void button1_Click(object sender, System.EventArgs e)
{label2.Visible=true;radioButton1.Visible=true;radioButton2.Visible=true;}

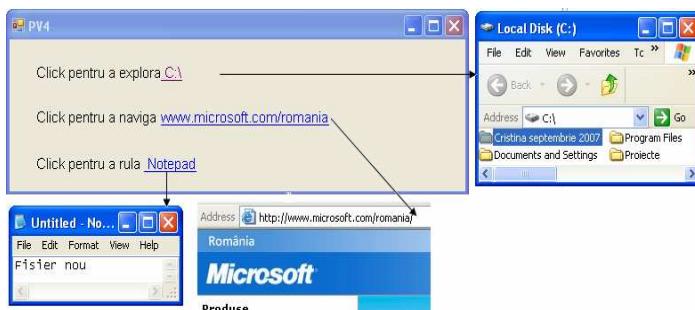
```



Exercitiu (Test grilă) Construiți un test grilă care conține 5 itemi cu câte 4 variante de răspuns (alegere simplă sau multiplă), memorăți răspunsurile date și afișați, după efectuarea testului, într-o casetă text, în dreptul fiecărui item, răspunsul corect.

- **LinkLabel** afișează un text cu posibilitatea ca anumite părți ale textului (**LinkArea**) să fie desenate ca și hyperlink-uri. Pentru a face link-ul funcțional trebuie tratat evenimentul **LinkClicked**.

În exemplul **PV4**, prima etichetă permite afișarea conținutului discului C:, a doua legătură este un link către pagina www.microsoft.com/romania și a treia accesează Notepad.



```

private void etichetaC_LinkClicked ( object sender,
                                    LinkLabelLinkClickedEventArgs e )
{ etichetaC.LinkVisited = true;
  System.Diagnostics.Process.Start( @"C:\" );
}
private void etichetaI_LinkClicked( object sender,
                                    LinkLabelLinkClickedEventArgs e )
{ etichetaI.LinkVisited = true;
  System.Diagnostics.Process.Start( "IEExplore",
                                    "http://www.microsoft.com/romania/" );
}
private void etichetaN_LinkClicked( object sender,
                                    LinkLabelLinkClickedEventArgs e )
{ etichetaN.LinkVisited = true;
  System.Diagnostics.Process.Start( "notepad" );
}

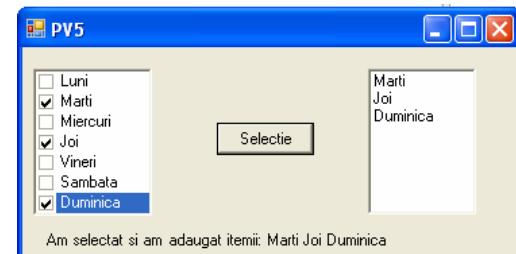
```

Exercițiu (Memorator) Construiți o aplicație care să conțină patru legături către cele patru fișiere/ pagini care conțin rezumatul capitolelor studiate.

- Controale pentru listare (**ListBox**, **CheckedListBox**, **ComboBox**, **ImageList**) ce pot fi legate de un **DataSet**, de un **ArrayList** sau de orice tablou (orice sursă de date ce implementează interfața **IEnumerable**).

În exemplul **PV5** elementele selectate din **CheckedListBox** se adaugă în **ListBox**. După adăugarea pe formular a **CheckedListBox**-ului, stabilim colecția de itemi (Properties-Items-Collection), butonul **Selectie** și **ListBox**-ul.

Evenimentul **Click** asociat butonului **Selectie** golește mai întâi **listBox**-ul (**listBox1.Items.Clear()**) și după aceea adaugă în ordine fiecare element selectat din **CheckedListBox**. Suplimentar se afișează o etichetă cu itemii selectați.



```

void button1_Click(object source, System.EventArgs e)
{
  String s = "Am selectat si am adaugat itemii: ";
  listBox1.Items.Clear();
  foreach ( object c in checkedListBox1.CheckedItems )
  { listBox1.Items.Add(c);
    s = s + c.ToString();s = s + " ";
  }
  label1.Text = s;
}

```

Exercițiu (Filtru) Construiți o aplicație care afișează fișierele dintr-un folder ales care au un anumit tip (tipul fișierelor este ales de utilizator pe baza unui **CheckedListBox**)

Aplicația **PV6** este un exemplu de utilizare a controlului **ImageList**. Apăsarea butonului **Desene** va adăuga fișierele *.gif din folderul **C:\Imagini** în listă și va afișa conținutul acestora. Butonul **Animate** va determina afișarea fișierelor *.gif cu ajutorul **PictureBox**.



```

ImageList desene_animate = new System.Windows.Forms.ImageList();
private void construieste_lista_Click(object sender, System.EventArgs e)
{
    // Configureaza lista
    desene_animate.ColorDepth = System.Windows.Forms.ColorDepth.Depth8Bit;
    desene_animate.ImageSize = new System.Drawing.Size(60, 60);
    desene_animate.Images.Clear();
    string[] gif_uri = Directory.GetFiles("C:\\\\Imagini", "*.gif");
    // se construieste un obiect Imagine pentru fiecare fisier si se adauga la ImageList.

    foreach (string fisier_gif in gif_uri)
    {Bitmap desen= new Bitmap (fisier_gif);
    desene_animate.Images.Add(desen);pictureBox2.Image=desen; }

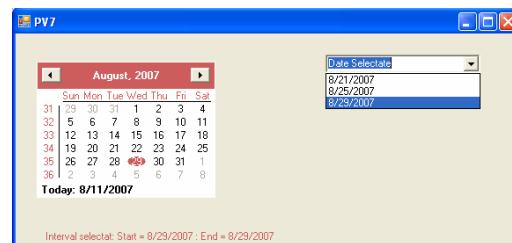
    Graphics g = this.CreateGraphics();
    // Deseneaza fiecare imagine utilizand metoda ImageList.Draw()
    for (int i = 0; i < desene_animate.Images.Count; i++)
        desene_animate.Draw(g, 60 + i * 60, 60, i);
    g.Dispose();
}

```

Exercițiu (Thumbnails) Afișați într-o fereastră conținutul folder-ului curent în mod View- Thumbnails.

- **MonthCalendar** afișează un calendar prin care se poate selecta o dată (zi, luna, an) în mod grafic. Proprietățile mai importante sunt: **MinDate**, **MaxDate**, **TodayDate** ce reprezintă data minimă/maximă selectabilă și data curentă (care apare afișată diferențiat sau nu în funcție de valorile proprietăților **ShowToday**, **ShowTodayCircle**). Există 2 evenimente pe care controlul le expune: **DateSelected** și **DateChanged**. În rutinele de tratare a acestor evenimente, programatorul are acces la un obiect de tipul **DateRangeEventArgs** care conține proprietățile **Start** și **End** (reprazentând intervalul de timp selectat).

Formularul din aplicația **PV7** conține un calendar pentru care putem selecta un interval de maximum 30 de zile, sunt afișate săptămânile și ziua curentă. Intervalul selectat se afișează prin intermediul unei etichete. Dacă se selectează o dată atunci aceasta va fi adăugată ca item într-un ComboBox (orice dată poate apărea cel mult o dată în listă).



După adăugarea celor 3 controale pe formular, stabilim proprietățile pentru **monthCalendar1** (**ShowWeekNumber=True**, **MaxSelectionCount=30**, etc.) și precizăm ce se execută atunci când selectăm un interval de timp:

```

private void monthCalendar1_DateSelected(object sender,
                                         System.Windows.Forms.DateRangeEventArgs e)
{
    this.label1.Text = "Interval selectat: Start = "
                      + e.Start.ToShortDateString() + " : End = "
                      + e.End.ToShortDateString();

    if (e.Start.ToShortDateString() == e.End.ToShortDateString())
    {
        String x = e.Start.ToShortDateString();
        if (!(comboBox1.Items.Contains(x)))
            comboBox1.Items.Add(e.End.ToShortDateString());
    }
}

```

- **DateTimePicker** este un control care (ca și MonthCalendar) se poate utiliza pentru a selecta o dată. La click se afișează un control de tip MonthCalendar, prin care se poate selecta data dorită. Fiind foarte asemănător cu MonthCalendar, proprietățile prin care se poate modifica comportamentul controlului sunt identice cu cele ale controlului MonthControl.

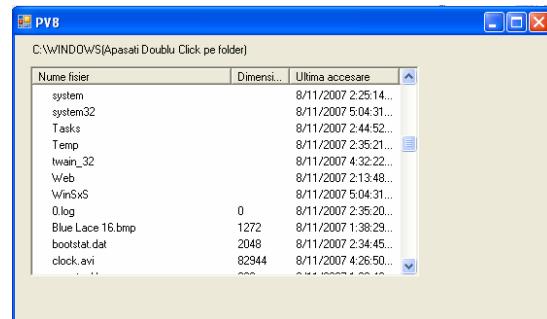
Exercițiu (Formular) Construiți un formular de introducere a datelor necesare realizării unei adrese de e-mail. Data nașterii va fi selectată direct utilizând MonthCalendar.

- **ListView** este folosit pentru a afișa o colecție de elemente în unul din cele 4 moduri (**Text**, **Text+Imagini mici**, **Imagini mari**, **Detalii**). Acesta este similar grafic cu ferestrele în care se afișează fișierele dintr-un anumit director din Windows Explorer. Fiind un control complex, conține foarte multe proprietăți, printre care:
View (selectează modul de afișare (LargeIcon, SmallIcon, Details, List)), **LargeImageList**, **SmallImageList** (icon-urile de afișat în modurile LargeIcon, SmallIcon), **Columns**(utilizat doar în modul Details, pentru a defini coloanele de afișat), **Items**(elementele de afișat).

Aplicația **PV8** este un exemplu de utilizare **ListView**. Se pornește de la rădăcină și se afișează conținutul folder-ului selectat cu dublu click. La expandare se afișează numele complet, data ultimei accesări și, în cazul fișierelor, dimensiunea.

Controlul **lista_fisiere** este de tip **ListView**.

Functia **ConstruiesteHeader** permite stabilirea celor trei coloane de afișat.



Nume fisier	Dimensiune	Ultima accesare
system		8/11/2007 2:25:14...
system32		8/11/2007 5:04:31...
Tasks		8/11/2007 2:44:52...
Temp		8/11/2007 2:35:21...
tware_32		8/11/2007 4:32:22...
Web		8/11/2007 2:13:48...
WinSxS		8/11/2007 5:04:31...
0.log	0	8/11/2007 2:35:20...
Blue Lace 16.bmp	1272	8/11/2007 1:38:29...
bootstat.dat	2048	8/11/2007 2:34:45...
clock.avi	82944	8/11/2007 4:26:50...
...

```
private void ConstruiesteHeader()
{
    ColumnHeader colHead; colHead = new ColumnHeader();
    colHead.Text = "Nume fisier";
    this.lista_fisiere.Columns.Add(colHead);
    colHead = new ColumnHeader(); colHead.Text = "Dimensiune";
    this.lista_fisiere.Columns.Add(colHead);
    colHead = new ColumnHeader(); colHead.Text = "Ultima accesare";
    this.lista_fisiere.Columns.Add(colHead);
}
```

Pentru item-ul selectat se afișează mai întâi folderele și după aceea fișierele. Pentru aceasta trebuie să determinăm conținutul acestuia:

```
ListViewItem lvi;
ListViewItem.ListViewSubItem lvsI;
this.calea_curenta.Text = radacina + "(Doublu Click pe folder)";
System.IO.DirectoryInfo dir = new System.IO.DirectoryInfo(radacina);
DirectoryInfo[] dirs = dir.GetDirectories();
FileInfo[] files = dir.GetFiles();
```

să ștergem vechiul conținut al listei:

```
this.lista_fisiere.Items.Clear();
this.lista_fisiere.BeginUpdate();
```

și să adăugăm fiecare nou item (coloana a doua este vidă în cazul foldere-lor):

```
foreach (System.IO.DirectoryInfo fi in dirs)
{
    lvi = new ListViewItem(); lvi.Text = fi.Name;
    lvi.ImageIndex = 1; lvi.Tag = fi.FullName;
    lvsI = new ListViewItem.ListViewSubItem();
    lvsI.Text = ""; lvi.SubItems.Add(lvsI);
    lvsI = new ListViewItem.ListViewSubItem();
    lvsI.Text = fi.LastAccessTime.ToString();
    lvi.SubItems.Add(lvsI); this.lista_fisiere.Items.Add(lvi);
}
```

Exercițiu (Ordonare) Modificați aplicația anterioară astfel încât apăsarea pe numele unei coloane să determine afișarea informațiilor ordonate după criteriul specificat (nume, dimensiune, data).

- Controle "de control" al executării (Timer) sau de dialog (**OpenFileDialog**, **SaveFileDialog**, **ColorDialog**, **FontDialog**, **ContextMenu**).

- Grupuri de controale **Toolbar** (*ToolStrip*) afișează o bară de butoane în partea de sus a unui formular. Se pot introduce vizuale butoane (printr-un designer, direct din Visual Studio.NET IDE), la care se pot seta atât textul afișat sau imaginea. Evenimentul cel mai util al acestui control este **ButtonClick** (care are ca parametru un obiect de tip **ToolBarButtonClickEventArgs**, prin care programatorul are acces la butonul care a fost apasat).

În aplicația următoare **PV9** cele 3 butoane ale toolbar-ului permit modificarea proprietăților textului introdus în casetă. Toolbar-ul se poate muta fără a depăși spațiul ferestrei. Schimbarea fontului se realizează cu ajutorul unui control **FontDialog()**, iar schimbarea culorii utilizează **ColorDialog()**

```
FontDialog fd = new FontDialog();
fd.ShowColor = true; fd.Color = Color.IndianRed;
fd.ShowApply = true;
fd.Apply += new EventHandler(ApplyFont);
if(fd.ShowDialog() != System.Windows.Forms.DialogResult.Cancel)
{
    this.richTextBox1.Font = fd.Font;
    this.richTextBox1.ForeColor = fd.Color;
}
ColorDialog cd = new ColorDialog();
cd.AllowFullOpen = true; cd.Color = Color.DarkBlue;
if(cd.ShowDialog() == System.Windows.Forms.DialogResult.OK)
    this.richTextBox1.ForeColor = cd.Color;
```

Mutarea toolbar-ului este dirijată de evenimentele produse atunci când apăsăm butonul de mouse și/sau ne deplasăm pe suprafața ferestrei.

```
private void toolBar1_MouseDown(object sender, MouseEventArgs e)
{
    // am apasat butonul de mouse pe toolbar
    am_apasat = true;
    forma_deplasata = new Point(e.X, e.Y); toolBar1.Capture = true;
}
private void toolBar1_MouseUp(object sender, MouseEventArgs e)
{
    am_apasat = false; toolBar1.Capture = false;
}
private void toolBar1_MouseMove(object sender, MouseEventArgs e)
{
    if (am_apasat)
    {
        if (toolBar1.Dock == DockStyle.Top || toolBar1.Dock == DockStyle.Left)
            { // dacă depăsește atunci duc în stanga sus
                if (forma_deplasata.X < (e.X - 20) || forma_deplasata.Y < (e.Y - 20))
                    { am_apasat = false; // Disconect toolbar
                        toolBar1.Dock = DockStyle.None;
                        toolBar1.Location = new Point(10, 10);
                        toolBar1.Size = new Size(200, 45);
                        toolBar1.BorderStyle = BorderStyle.FixedSingle;
                    }
            }
        else if (toolBar1.Dock == DockStyle.None)
            { toolBar1.Left = e.X + toolBar1.Left - forma_deplasata.X;
                toolBar1.Top = e.Y + toolBar1.Top - forma_deplasata.Y;
                if (toolBar1.Top < 5 || toolBar1.Top > this.Size.Height - 20)
                    { am_apasat = false; toolBar1.Dock = DockStyle.Top;
                        toolBar1.BorderStyle = BorderStyle.Fixed3D;
                    }
                else if (toolBar1.Left < 5 || toolBar1.Left > this.Size.Width - 20)
                    { am_apasat = false; toolBar1.Dock = DockStyle.Left;
                        toolBar1.BorderStyle = BorderStyle.Fixed3D;
                    }
            }
    }
}
```

Exercițiu (Editor) Realizați un editor de texte care conține un control toolbar cu butoanele uzuale.



- **Controale container (GroupBox, Panel, TabControl)** sunt controale ce pot conține alte controale.

Aplicația **PV10** simulează lansarea unei comenzi către un magazin de jucării. Se utilizează 4 pagini de Tab pentru a simula selectarea unor opțiuni ce se pot grupa pe categorii.

Exercițiu (Magazin) Dezvoltați aplicația precedentă astfel încât pe o pagină să se afișeze modelele disponibile (Imagine+detalii) și să se permită selectarea mai multor obiecte. Ultima pagină reprezintă coșul de cumpărături.



- **Grupuri de controale tip Meniu (MenuStrip, ContextMenuStrip etc.)**

Un formular poate afișa un singur meniu principal la un moment dat, meniul asociat inițial fiind specificat prin proprietatea **Form.MainMenuStrip**. Meniul care este afișat de către un formular poate fi schimbat dinamic la rulare :

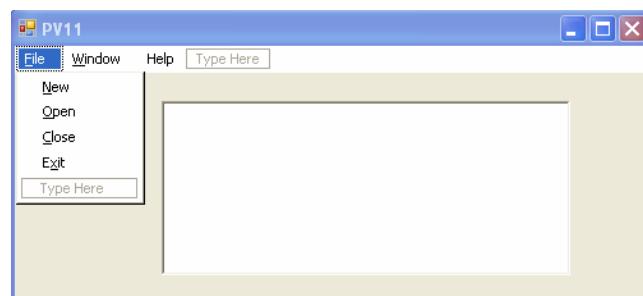
```
switch(cond) { case cond1:this.MainMenuStrip = this.mainMenu1;break;
               case cond2:this.MainMenuStrip = this.mainMenu2;
}
```

unde **mainMenu1** și **mainMenu2** sunt obiecte de tip **MenuStrip**. Editarea unui astfel de obiect se poate face utilizând **Menu Designer**. Clasa **MenuStrip** are o colecție de **MenuItem** care conține 0 sau mai multe obiecte de tip **MenuItem**. Fiecare dintre aceste obiecte de tip **MenuItem** are 0 sau mai multe obiecte de tip **MenuItem**, care vor constitui noul nivel de itemi (Ex: File → New, Save, Open, Close, Exit).

Proprietățile Checked si RadioCheck indică itemul selectat, **Enabled** and **Visible** determină dacă un item poate fi sau nu selectat sau vizibil, **Shortcut** permite asignarea unei combinații de taste pentru selectarea unui item al meniului și **Text** memorează textul care va fi afișat pentru respectivul item al meniului.

Evenimentul **Click** are loc când un utilizator apasă un item al meniului.

Exemplul **PV11** permite, prin intermediul unui meniu, scrierea unui fisier Notepad, afișarea continutului acestuia într-o casetă text, schimbarea fontului și colorii de afișare, ștergerea conținutului casetei, afișarea unor informații teoretice precum și Help dinamic. Au fost definite chei de acces rapid pentru accesarea componentelor meniului.



File → New permite scrierea unui fișier notepad nou

```
System.Diagnostics.Process.Start( "notepad" );
```

File → Open selectează și afișează în caseta text conținutul unui fișier text.

```
OpenFileDialog of = new OpenFileDialog();
of.Filter = "Text Files (*.txt)|*.txt";
of.Title = "Fisiere Text";
if (of.ShowDialog() == DialogResult.Cancel) return;
richTextBox1.Text= "";richTextBox1.Visible=true;
FileStream strm;
try{strm = new FileStream (of.FileName, FileMode.Open, FileAccess.Read);
StreamReader rdr = new StreamReader (strm);
while (rdr.Peek() >= 0){string str = rdr.ReadLine ();
richTextBox1.Text=richTextBox1.Text+ " "+str;}
}
catch (Exception){MessageBox.Show ( "Error opening file", "File Error",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);}
```

File → Close șterge conținutul casetei text, File → Exit închide aplicația

Window → Font și Window → Color permit stabilirea fontului/colorii textului afișat.

```
Help→ DinamicHelp accesează System.Diagnostics.Process.Start("IExplore",
    "http://msdn2.microsoft.com/en-us/default.aspx");
```

Help→ About PV afișează în caseta text informații despre implementarea unui menu.

Exercițiu (Fisiere) Construți un menu care să permită efectuarea operațiilor uzuale cu fișiere.

4.5. System.Drawing

Spațiul `System.Drawing` conține tipuri care permit realizarea unor desene 2D și au rol deosebit în proiectarea interfețelor grafice.

Un obiect de tip `Point` este reprezentat prin coordonatele unui punct într-un spațiu bidimensional (exemplu: `Point myPoint = new Point(1, 2);`)

Point este utilizat frecvent nu numai pentru desene, ci și pentru a identifica în program un punct dintr-un anumit spațiu. De exemplu, pentru a modifica poziția unui buton în fereastră putem asigna un obiect de tip `Point` proprietății `Location` indicând astfel poziția colțului din stânga-sus al butonului (`button.Location = new Point(100, 30)`). Putem construi un obiect de tip `Point` pentru a redimensiona un alt obiect.

```
Size mySize = new Size(15, 100);
Point myPoint = new Point(mySize);
System.Console.WriteLine("X: " + myPoint.X + ", Y: " + myPoint.Y);
```

Structura `Color` conține date, tipuri și metode utile în lucrul cu culori. Fiind un tip valoare (`struct`) și nu o clasă, aceasta conține date și metode, însă nu permite instantiere, constructori, deconstructor, moștenire.

```
Color myColor = Color.Brown; button1.BackColor = myColor;
```

Substructura `FromArgb` a structurii `Color` returnează o culoare pe baza celor trei componente ale oricărei culori (red, green, blue).

Clasa `Graphics` este o clasă sigilată reprezentând o arie rectangulară care permite reprezentări grafice. De exemplu, o linie frântă se poate realiza astfel:

```
Point[] points = new Point[4];
points[0] = new Point(0, 0); points[1] = new Point(0, 120);
points[2] = new Point(20, 120); points[3] = new Point(20, 0);
Graphics g = this.CreateGraphics();
Pen pen = new Pen(Color.Yellow, 2);
g.DrawLine(pen, points);
```



Aplicația **PV12** este un exercițiu care desenează cercuri de raze și culori aleatoare și emite sunete cu frecvență aleatoare.

```
Random x = new Random();
Console.Beep(300 + x.Next(1000), 150);
Graphics g = e.Graphics;
i = 1 + x.Next(30);
p=new Pen(System.Drawing.Color.FromArgb(x.Next(256),x.Next(256),x.Next(256)));
g.DrawEllipse(p, x.Next(100), x.Next(100), i, i);
Console.Sleep(200);
```



În exemplul **PV13** se construiește o pictogramă pe baza unei imagini.

```
Image thumbnail;
private void Thumbnails_Load(object sender, EventArgs e)
{ try{Image img = Image.FromFile("C:\\\\Imagini\\\\catel.jpg");
    int latime=100, inaltime=100;
    thumbnail=img.GetThumbnailImage(latime, inaltime,null,
    IntPtr.Zero);}
    catch{MessageBox.Show("Nu există fisierul");}
}
private void Thumbnails_Paint(object sender, PaintEventArgs e)
{e.Graphics.DrawImage(thumbnail, 10, 10);}
```

4.6. Validarea informațiilor de la utilizator

Înainte ca informațiile de la utilizator să fie preluate și transmise către alte clase, este necesar să fie validate. Acest aspect este important, pentru a preveni posibilele erori. Astfel, dacă utilizatorul introduce o valoare reală (*float*) când aplicația așteaptă un întreg (*int*), este posibil ca aceasta să se comporte neprevăzut abia câteva secunde mai târziu, și după multe apeluri de metode, fiind foarte greu de identificat cauza primară a problemei.

Validarea la nivel de câmp

Datele pot fi validate pe măsură ce sunt introduse, asociind o prelucrare unuia dintre handlelele asociate evenimentelor la nivel de control (*Leave*, *TextChanged*, *MouseUp* etc.)

```
private void textBox1_KeyUp(object sender,
                           System.Windows.Forms.KeyEventArgs e)
{
    if(e.Alt==true) MessageBox.Show("Tasta Alt e apasata"); // sau
    if(Char.IsDigit(e.KeyChar)==true) MessageBox.Show("Ati apasat o cifra");
}
```

Validarea la nivel de utilizator

În unele situații (de exemplu atunci când valorile introduse trebuie să se afle într-o anumită relație între ele), validarea se face la sfârșitul introducerii tuturor datelor la nivelul unui buton final sau la închiderea ferestrei de date.

```
private void btnValidate_Click(object sender, EventArgs e)
{
    foreach(System.Windows.Forms.Control a in this.Controls)
    {
        if( a is System.Windows.Forms.TextBox & a.Text=="")
            { a.Focus();return;}
    }
}
```

ErrorProvider

O manieră simplă de a semnala erori de validare este aceea de a seta un mesaj de eroare pentru fiecare control.

```
myErrorProvider.SetError(txtName, " Numele nu are spatii in stanga");
```

Aplicatii recapitulative.

Urmăriți aplicațiile și precizați pentru fiecare dintre ele controalele utilizate, evenimentele tratate: Forma poloneza (PV14), Triunghi (PV15), Ordonare vector(PV16), Subsir crescător de lungime maximă(PV17), Jocul de Nim (PV18)

Exercițiu (Test grila) Realizați un generator de teste grilă (întrebările sunt preluate dintr-un fisier text, pentru fiecare item se precizează punctajul, enunțul, răspunsul corect, distractorii și o imagine asociată enunțului (dacă există). După efectuarea testului se afișează rezultatul obținut și statistica răspunsurilor.

5. Aplicații orientate pe date

5.1. Structuri de date

Construirea unei aplicații ce gestionează un volum mare de date necesită o atenție particulară privind organizarea acestor date. Sursele de date trebuie să fie integrate corespunzător într-o aplicație OOP. Faptul că o stivă din clasa *Stack* este implementată ca vector sau nu, este de mică importanță pentru programatorul pe platformă .NET. Dar pentru operațiile executate de o aplicație ce monitorizează traficul dintr-o gară de triaj, este important de știut dacă trebuie memorat sau nu un istoric al acestor operații, dacă sunt folosite informațiile de acest tip în prelucrări ulterioare sau sunt doar listate sub formă de rapoarte periodice, etc. Dacă datele trebuie exportate către alte aplicații, atunci se pune problema formatului în care vor fi salvate. Pentru colecțiile consistente de date care suferă prelucrări ample și frecvente se pune problema suportului de memorare astfel încât să suporte tehnici de acces rapide etc.

5.2. Colecții de date

Colecția, în general, reprezintă un ansamblu bine structurat de componente de același tip, ansamblu ce permite identificarea rapidă a oricărei componente din colecție. Definiția este aplicabilă și colecțiilor de date. Uneori, cloecțiile de date sunt percepute ca date *externe* (aflate pe harddisc sau alte suporturi de memorare), dar asta nu exclude posibilitatea organizării datelor *interne* sub forma unor colecții. În C#, colecțiile sunt clase specializate aparținând spațiului System.Collection. Despre colecții am mai vorbit în cadrul capitolului 3.11. Aducem unele completări în cele ce urmează vorbind despre **structuri generice de date**.

Atributul "generic" se referă la proprietatea de a referi o întreagă categorie de obiecte. Clasele, structurile, metodele, interfețele, delegați și metodele pot fi generice, adică pot fi concepute astfel încât să depindă de unul sau mai multe tipuri de date pe care acestea le memorează sau manipulează.

De exemplu, la declararea unei clase obișnuite, noi stabilim elementele fixe ale acesteia: numele și tipurile datelor, numele metodelor³³ și altor componente ce formează clasa respectivă. Pentru o funcție (metodă) obișnuită sunt definite atât numele, cât și tipul parametrilor funcției³⁴.

C# introduce, începând de la versiunea 2.0, parametrizarea tipurilor, adică posibilitatea declarării unor clase, metode etc. în care tipul datelor manevrate nu este cunoscut decât la apel. Acest tip de date constituie un parametru al clasei, metodei etc.

Vom defini în mod obișnuit și, paralel, în mod generic clasa Stiva (amintită și în capitolul 1.1) pentru a pune în evidență diferențele induse de modul de lucru ... generic:

<pre>class Stiva { int[] st = new int[10]; int vf; public Stiva() { vf = -1; } public void Push(int x) { st[++vf] = x; } public int Pop() { return st[vf--]; }³⁵ public void scrie() { for (int i = 0; i <= vf; i++) Console.WriteLine(st[i]); } }</pre>	<pre>class Stiva<T> { T[] st = new T[10]; int vf; public Stiva() { vf = -1; } public void Push(T x) { st[++vf] = x; } public T Pop()³⁵ { return st[vf--]; } public void scrie() { for (int i = 0; i <= vf; i++) Console.WriteLine(st[i]); } }</pre>
--	---

³³ A nu se confunda capacitatea de a defini mai multe metode cu același nume cu faptul că numele este bine determinat. A spune că numele nu este definit ar presupune un fel de declarare cu numele ... și abia la apel să se stabilească ce nume este scris în loc de ... ☺

³⁴ Numărul parametrilor este și el bine definit, deși funcțiile care au definit un parametru de tip tablou, permit apelul cu un număr variabil de parametri efectivi.

³⁵ Versiuni mai vechi de C nu execută conform așteptărilor returnul valorii cu postincrementare.

```

...
static void Main(string[] args)
{
    Stiva s = new Stiva();
    s.Push(7); s.Push(5); s.scrie();
    Console.WriteLine("Ex{0}", s.Pop());
    s.scrie();
    s.Push(7); s.scrie();
    Console.WriteLine("Ex{0}", s.Pop());
    s.scrie();
}
...
static void Main(string[] args)
{
    Stiva<int> s = new Stiva<int>();
    s.Push(7); s.Push(5); s.scrie();
    Console.WriteLine("Ex{0}", s.Pop());
    s.scrie();
    s.Push(7); s.scrie();
    Console.WriteLine("Ex{0}", s.Pop());
    s.scrie();
}

```

Dintre clasele generice "predefinite", cele mai des utilizate sunt colecțiile generice. Astfel, clasele și interfețele din spațiul **System.Collection.Generic** permit organizarea datelor proprii, indiferent de tipul acestora, în structuri specifice cum ar fi: liste (*List*), liste dublu înlăntuite (*LinkedList*), stive (*Stack*), cozi (*Queue*), dicționare (*Dictionary*) etc.

De exemplu, dorim să gestionăm un ansamblu de ferestre asemănătoare, instantiată dintr-o aceeași clasă **MyForm** (să zicem) derivată din **System.Windows.Forms.Form**. Putem memora în program sau ca element static al clasei MyForm o structură de tip listă:

```
List<MyForm> fer=new List<MyForm>()
```

La crearea oricărei ferestre *f* de tipul **MyForm** (*MyForm f=new MyForm(); f.Show();*) se adaugă referința ferestrei în lista *fer* (*fer.Add(f)*). Analog, la închiderea ferestrei, se va elimina referința acesteia din listă (*fer.Remove(f)*).

Exercițiu: Presupunem că implementăm o aplicație în care se completează datele unui pacient într-o fereastră. Pentru a nu încărca ferestra curentă, aceasta este proiectată să conțină butoane ce permit deschiderea unor ferestre secundare pentru completarea unor subcategorii de date (istoricul suferințelor cronice, analize efectuate, adresa detaliată etc.). De exemplu, în fereastră se completează numele pacientului, iar apoi, apăsând pe butonul Adresa, se deschide o fereastră ce permite completarea câmpurilor ce formează adresa, la închiderea ferestrei revenind la fereastra principală. Desigur, și o fereastră secundară poate avea la rândul ei butoane pentru alte ferestre secundare în raport cu aceasta. Stabilită că tip de structură generică se poate utiliza pentru gestiunea ferestrelor deschise la un moment dat³⁶.

5.3. ADO.NET

ADO.NET (*ActiveX Data Objects*) reprezintă o parte componentă a nucleului .NET Framework ce permite accesarea și manipularea datelor. Amintim că o sursă de date poate fi: un fișier text, un fișier Excel sau XML, o bază de date Dbase, Access, SQL etc. Lucrul cu o sursă de date se poate face fie conectat, fie deconectat de la sursa de date. ADO.NET implementează clase ce oferă servicii atât pentru lucrul în stil deconectat cât și conectat, oferă instrumentele de utilizare și reprezentare XML, de combinare a datelor din diferite surse și de diferite tipuri (pe bază mecanismelor de reprezentare comună implementate de .NET). Maniera de lucru deconectată recomandă metoda ca fiind mai eficientă în proiectarea aplicațiilor pentru Internet decât alte tehnologii cum ar fi ADO sau ODBC.

Deoarece există mai multe tipuri de baze de date e nevoie de câte o bibliotecă specializată de clase și interfețe care să implementeze un "protocol" specific pentru fiecare.

5.4. Conectarea la o sursă de date

Înainte de orice operație cu o sursă de date externă, trebuie realizată o conexiune (legătură) cu acea sursă. Clasele din categoria Connection (*SQLConnection*, *OleDbConnection* etc.) conțin date referitoare la sursa de date (locația, numele și parola contului de acces, etc.), metode pentru deschiderea/inchiderea conexiunii, pornirea unei tranzacții etc. Aceste clase se găsesc în subspații (*SqlClient*, *OleDb* etc.) ale spațiului **System.Data**. În plus, ele implementează interfața *IDbConnection*.

³⁶ punem accentul pe faptul că ferestrele sunt *modale*

Pentru deschiderea unei conexiuni prin program se poate instanția un obiect de tip conexiune, precizându-i ca parametru un sir de caractere conținând date despre conexiune. Dăm două exemple de conectare la o sursă de date SQL respectiv Access:

```
using System.Data.SqlClient;
...
SqlConnection co = new SqlConnection(@"Data Source=serverBD;
Database=scoala;User ID=elev;Password=madonna");
co.Open();
...
using System.Data.OleDb;
...
OleDbConnection con =
    new OleDbConnection(@"Provider=Microsoft.Jet.OLEDB.4.0;
        Data Source=C:\Date\scoala.mdb");
cn.Open();
...
```

Proprietățile unei conexiuni

ConnectionString (String): conține un string cu o succesiune de parametri de forma parametru=valoare, despărțiti prin ;. Parametrii pot fi:

- **provider**: specifică furnizorul de date pentru conectarea la sursa de date. Acest furnizor trebuie precizat doar dacă se folosește OLE DB .NET Data Provider, și nu se specifică pentru conectare la SQL Server
- **Data Source**: se specifică numele serverului de baze de date sau numele fișierului de date
- **Initial Catalog (Database)**: specifică numele bazei de date. Baza de date trebuie să se găsească pe serverul dat în Data Source.
- **User ID**: specifică un nume de cont de utilizator care are acces de logare la server.
- **Password**: specifică parola contului de mai sus.

ConnectionTimeout (int): specifică numărul de secunde pentru care un obiect de conexiune poate să aștepte pentru realizarea conectării la server înainte de a se genera o excepție. (implicit 15). Se poate specifica o valoare diferită de 15 în ConnectionString folosind parametrul Connect Timeout. Valoarea Timeout=0 specifică așteptare nelimitată.

```
SqlConnection cn = new SqlConnection("Data Source=serverBD;
Database=scoala;User ID=elev;Password=madonna; Connect Timeout=30");
```

Database (string): returnează numele bazei de date la care s-a făcut conectarea. Este necesară pentru a arăta unui utilizator care este baza de date pe care se face operarea

Provider (string): returnează furnizorul

ServerVersion (string): returnează versiunea de server la care s-a făcut conectarea.

State (enumerare de componente *ConnectionState*): returnează starea curentă a conexiunii. Valorile posibile: *Broken*, *Closed*, *Connecting*, *Executing*, *Fetching*, *Open*.

Metodele unei conexiuni

Open(): deschide o conexiune la baza de date

Close() și **Dispose()**: închid conexiunea și eliberează toate resursele alocate pentru ea

BeginTransaction(): pentru executarea unei tranzacții pe baza de date; la sfârșit se apelează **Commit()** sau **Rollback()**.

ChangeDatabase(): se modifică baza de date la care se vor face conexiunile. Noua bază de date trebuie să existe pe același server ca și precedenta.

CreateCommand(): creează o comandă (un obiect de tip *Command*) validă asociată conexiunii curente.

Evenimentele unei conexiuni

StateChange: apare atunci când se schimbă starea conexiunii. Handlerul corespunzător (de tipul delegat *StateChangeEventHandler*) spune între ce stări s-a făcut tranziția.

InfoMessage: apare când furnizorul trimite un avertisment sau un mesaj către client.

5.5. Executarea unei comenzi SQL

Clasele din categoria Command (*SQLCommand*, *OleDbCommand* etc.) conțin date referitoare la o comandă SQL (SELECT, INSERT, DELETE, UPDATE) și metode pentru executarea unei comenzi sau a unor proceduri stocate. Aceste clase implementează interfața *IDbCommand*. Ca urmare a interogării unei baze de date se obțin obiecte din categoriile *DataReader* sau *DataSet*. O comandă se poate executa numai după ce s-a stabilit o conexiune cu baza de date corespunzătoare.

```
SqlConnection co = new SqlConnection(@"Data Source=serverBD;
Database=scoala;User ID=elev;Password=madonna");
co.Open();
SqlCommand cmd = new SqlCommand("SELECT * FROM ELEVI", con);
```

Proprietățile unei comenzi

CommandText (String): conține comanda SQL sau numele procedurii stocate care se execută pe sursa de date.

CommandTimeout (int): reprezintă numărul de secunde care trebuie să fie așteptat pentru executarea comenzi. Dacă se depășeste acest timp, atunci se generează o excepție.

CommandType (enumerare de componente de tip *CommandType*): reprezintă tipul de comandă care se execută pe sursa de date. Valorile pot fi: *StoredProcedure* (apel de procedură stocată), *Text* (comandă SQL obișnuită), *TableDirect* (numai pentru *OleDb*)

Connection (System. Data. [Provider].PrefixConnection): conține obiectul de tip conexiune folosit pentru legarea la sursa de date.

Parameters (System.Data.[Provider].PrefixParameterCollection): returnează o colecție de parametri care s-au transmis comenzi;

Transaction (System.Data.[Provider].PrefixTransaction): permite accesul la obiectul de tip tranzație care se cere a fi executat pe sursa de date.

Metodele unei comenzi

Constructori: *SqlCommand()* sau *SqlCommand(string CommandText)* sau

```
SqlCommand(string CommandText, SqlConnection con ) sau
SqlCommand(string CommandText,SqlConnection con,SqlTransaction trans)
```

Cancel() oprește o comandă aflată în executare.

Dispose() distrugе obiectul comandă.

ExecuteNonQuery() execută o comandă care nu returnează un set de date din baza de date; dacă comanda a fost de tip *INSERT*, *UPDATE*, *DELETE*, se returnează numărul de înregistrări afectate.

```
SqlCommand cmd = new SqlCommand();
cmd.CommandText = "DELETE FROM elevi WHERE nume = 'BARBU'";
cmd.Connection = con;
Console.WriteLine(cmd.ExecuteNonQuery().ToString()); //câte înreg.s-au șters
ExecuteReader() execută comanda și returnează un obiect de tip DataReader
SqlCommand cmd = new SqlCommand("SELECT * FROM elevi",con);
SqlDataReader reader = cmd.ExecuteReader();
while(reader.Read())
{ Console.WriteLine("{0} - {1}",
reader.GetString(0),reader.GetString(1));
}
reader.Close();
```

Metoda *ExecuteReader()* mai are un argument optional, *CommandBehavior* care descrie rezultatele și efectul asupra bazei de date: *CloseConnection* (conexiunea este închisă atunci când obiectul *DataReader* este închis), *KeyInfo* (returnează informație despre coloane și cheia primară), *SchemaOnly* (returnează doar informație despre coloane), *SequentialAccess* (pentru manevrarea valorilor binare cu *GetChars()* sau *GetBytes()*), *SingleResult* (se returnează un singur set de rezultate), *SingleRow* (se returnează o singură linie).

ExecuteScalar() execută comanda și returnează valoarea primei coloane de pe primul rând a setului de date rezultat; folosit pentru obținerea unor rezultate statistice

```
SqlCommand cmd = new SqlCommand("SELECT COUNT(*) FROM elevi", con);
SqlDataReader reader = cmd.ExecuteScalar();
Console.WriteLine(reader.GetString(0));
```

ExecuteXmlReader() returnează un obiect de tipul XmlReader obținut prin interogare
SqlCommand CMD=

```
new SqlCommand("SELECT * FROM elevi FOR XML MATE, EXAMEN", con);
System.Xml.XmlReader myXR = CMD.ExecuteXmlReader();
```

5.6. Seturi de date

Datele pot fi explorate în mod conectat (cu ajutorul unor obiecte din categoria *DataReader*), sau pot fi preluate de la sursă (dintr-un obiect din categoria *DataAdapter*) și înglobate în aplicația curentă (sub forma unui obiect din categoria *DataSet*).

Clasele **DataReader** permit parcurgerea într-un singur sens a sursei de date, fără posibilitate de modificare a datelor la sursă. Dacă se dorește modificarea datelor la sursă, se va utiliza ansamblul *DataAdapter + DataSet*.

Un obiect *DeatReader* nu are constructor, ci se obține cu ajutorul unui obiect de tip *Command* și prin apelul metodei *ExecuteReader()* (vezi exemplul de la pagina 41, jos). Evident, pe toată durata lucrului cu un obiect de tip *DataReader*, conexiunea trebuie să fie activă. Toate clasele *DataReader* (*SqlDataReader*, *OleDbDataReader* etc.) implementează interfața *IDataReader*.

Proprietăți:

IsClosed, *HasRows*, *Item* (indexator de câmpuri) și *FieldCount*

Metode:

Close() închidere obiectului și eliberarea resurselor; trebuie să preceadă închiderea conțiunii
GetBoolean(), *GetByte()*, *GetChar()*, *GetDateTime()*, *GetDecimal()*,
GetDouble(), *GetFloat()*, *GetInt16()*, *GetInt32()*, *GetInt64()*, *GetValue()*,
GetString() returnează valoarea unui câmp specificat, din înregistrarea curentă
GetBytes(), *GetChars()* citirea unor octeți/caractere dintr-un câmp de date binar
GetDataTypeName(), *GetName()* returnează tipul/numele câmpului specificat
IsDBNull() returnează true dacă în câmpul specificat prin index este o valoare NULL
NextResult() determină trecerea la următorul rezultat stocat în obiect (vezi exemplul)
Read() determină trecerea la următoarea înregistrare, returnând *false* numai dacă aceasta nu există; de reținut că inițial poziția curentă este **înaintea** primei înregistrări.

```
SqlCommand cmd=new SqlCommand("select * from elevi;select * from profi", conn );
conn.Open ();
SqlDataReader reader = cmd.ExecuteReader ();
do {
    while ( reader.Read () )
        {Console.WriteLine ("{0}\t{1}", reader[0], reader[1] );}
} while ( reader.NextResult () );
```

Folosirea combinată a obiectelor **DataAdapter** și **DataSet** permite operații de selectare, ștergere, modificare și adăugare la baza de date. Clasele *DataAdapter* generează obiecte care funcționează ca o interfață între sursa de date și obiectele *DataSet* interne aplicației, permitând prelucrări pe baza de date. Ele gestionează automat conexiunea cu baza de date astfel încât conexiunea să se facă numai atunci când este imperios necesar.

Un obiect *DataSet* este de fapt un set de tabele relaționate. Folosește serviciile unui obiect *DataAdapter* pentru a-și procura datele și trimită modificările înapoi către baza de date. Datele sunt stocate de un *DataSet* în format XML, același folosit și pentru transferul datelor.

În exemplul următor se preiau datele din tabelele elevi și profi:

```
SqlDataAdapter de=new SqlDataAdapter("SELECT nume,clasa FROM elevi, conn");
de.Fill(ds,"Elevi");//transferă datele în datasetul ds sub forma unei tabele locale numite elevi
SqlDataAdapter dp=new SqlDataAdapter("SELECT nume, clasdir FROM profi,conn");
dp.Fill(ds,"Profi");//transferă datele în datasetul ds sub forma unei tabele locale numite profi
```

Proprietățile unui DataAdapter

`DeleteCommand`, `InsertCommand`, `SelectCommand`, `UpdateCommand` (Command), conțin comenzi ce se execută pentru selectarea sau modificarea datelor în sursa de date. `MissingSchemaAction` (enumerare) determină ce se face atunci când datele aduse nu se potrivesc peste schema tablei în care sunt depuse. Poate avea următoarele valori:

- `Add` - implicit, DataAdapter adaugă coloana la schema tablei
- `AddWithKey` – se adaugă coloana și informații relativ la cheia primară
- `Ignore` - se ignoră lipsa coloanei respective, ceea ce duce la pierdere de date
- `Error` - se generează o excepție de tipul `InvalidOperationException`.

Metodele unui DataAdapter

Constructoril: `SqlDataAdapter();` sau `SqlDataAdapter(obiect_comanda);` sau `SqlDataAdapter(string_comanda, conexiune);`

`Fill()` permite umplerea unei tabele dintr-un obiect `DataSet` cu date. Permite specificarea obiectului `DataSet` în care se depun datele, eventual a numelui tablei din acest `DataSet`, numărul de înregistrare cu care să se înceapă popularea (prima având indicele 0) și numărul de înregistrări care urmează a fi aduse.

`Update()` permite transmiterea modificărilor efectuate într-un `DataSet` către baza de date.

Structura unui DataSet

Un `DataSet` este format din `Tables` (colecție formată din obiecte de tip `DataTable`; `DataTable` este compus la rândul lui dintr-o colecție de `DataRow` și `DataColumn`), `Relations` (colecție de obiecte de tip `DataRelation` pentru memorarea legăturilor părinte–copil) și `ExtendedProperties` ce conține proprietăți definite de utilizator.

Scenariul uzual de lucru cu datele dintr-o tabelă conține următoarele etape:

- popularea succesivă a unui `DataSet` prin intermediul unuia sau mai multor obiecte `DataAdapter`, apelând metoda `Fill` (vezi exemplul de mai sus)
- procesarea datelor din `DataSet` folosind numele tabelelor stabilite la umplere, `ds.Tables["elevi"]`, sau indexarea acestora, `ds.Tables[0]`, `ds.Tables[1]`
- actualizarea datelor prin obiecte comandă corespunzătoare operațiilor `INSERT`, `UPDATE` și `DELETE`. Un obiect `CommandBuilder` poate construi automat o combinație de comenzi ce reflectă modificările efectuate.

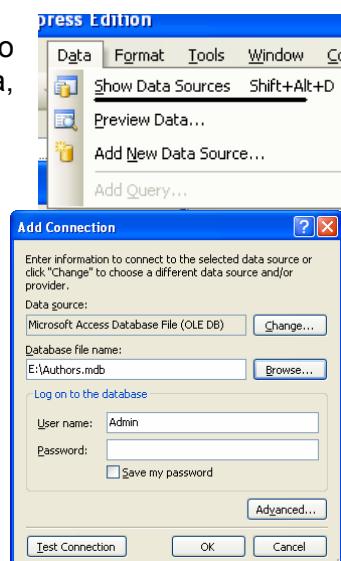
5.7. Proiectarea vizuală a seturilor de date

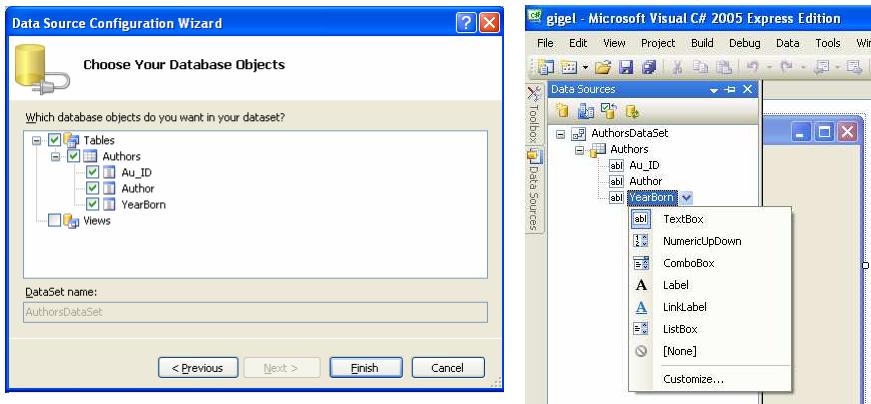
Mediul de dezvoltare Visual Studio dispune de instrumente puternice și sugestive pentru utilizarea bazelor de date în aplicații. Conceptual, în spatele unei ferestre în care lucrăm cu date preluate dintr-una sau mai multe tabele ale unei baze de date se află obiectele din categoriile `Connection`, `Command`, `DataAdapter` și `DataSet` prezentate. "La vedere" se află controale de tip `DataGridView`, sau `TableGridView`, `BindingNavigator` etc.

Meniul `Data` și fereastra auxiliară `Data Sources` ne sunt foarte utile în lucrul cu surse de date externe.

Să urmărim un scenariu de realizare a unei aplicații simple cu o fereastră în care putem vizualiza date dintr-o tabelă, putem naviga, putem modifica sau șterge înregistrări.

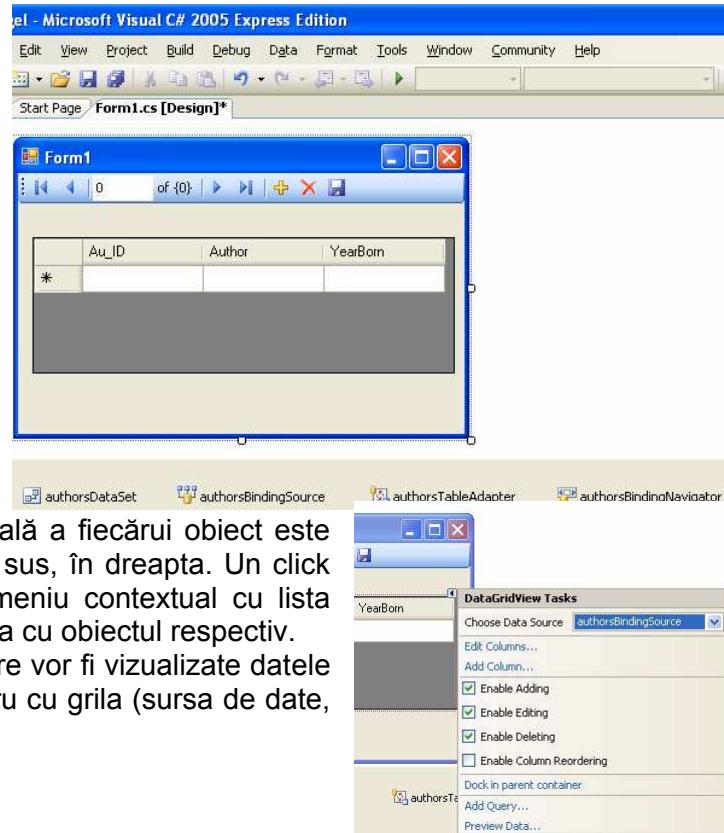
- Inițiem adăugarea unei surse de date (`Add New Source`)
- Configurăm cu atenție (asistați de "vrăjitor") conexiunea cu o sursă de tip SQL sau Access; figura surprinde elemente de conectare la o bază de date Access, numită `Authors`, bază stocată pe harddiscul local.
- Selectăm tabelele care ne interesează din baza de date și câmpurile din cadrul tabelei ce vor fi reținute în `TableAdapter` (din categoria `DataAdapter`)
- Când operațiunea se încheie, date relative la baza de date la care ne-am conectat sunt integrate în proiect și pictograma, ca și structura bazei de date apar în fereastra `Data Source`





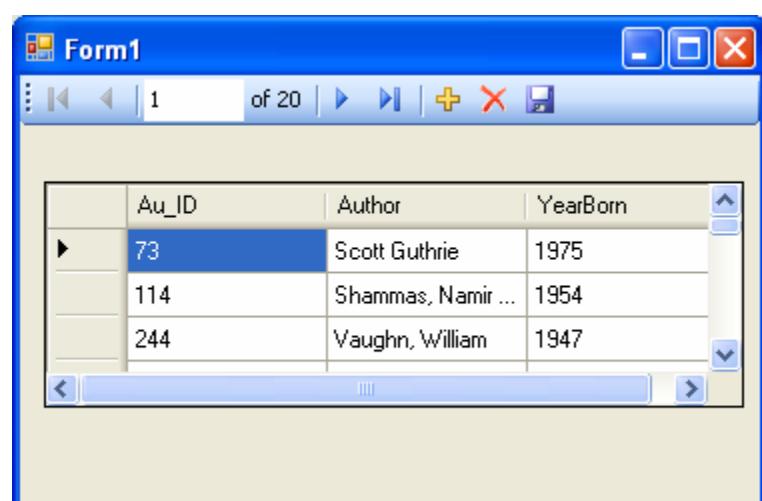
- Prin "tragerea" unor obiecte din fereastra *Data Sources* în fereastra noastră nouă, se creează automat obiecte specifice. În partea de jos a figurii se pot observa obiectele de tip *Dataset*, *TableAdapter*, *BindingSource*, *BindingNavigator* și, în fereastră, *TableGridView*.

BindingNavigator este un tip ce permite, prin instantiere, construirea barei de navigare ce permite operații de deplasare, editare, ștergere și adăugare în tabel.



Să observăm că reprezentarea vizuală a fiecărui obiect este înzestrată cu o săgeată în partea de sus, în dreapta. Un click pe această săgeată activează un meniu contextual cu lista principalelor operații ce se pot efectua cu obiectul respectiv. Meniul contextual asociat grilei în care vor fi vizualizate datele permite configurarea modului de lucru cu grila (sursa de date, operațiile permise și altele).

În timpul rulării aplicației, bara de navigare și elementele vizuale ale grilei permit operațiile de bază cu înregistrările bazei de date. Operațiile care modifică baza de date trebuie să fie definitivate prin salvarea noilor date .



Programarea Web cu ASP .Net 2.0

CUPRINS

CUPRINS	49
1. ARHITECTURI SOFTWARE	51
1.1. MODELUL CLIENT- SERVER.....	51
1.2. MODELUL MULTI-STRAT.....	52
1.3. APlicații ORIENTATE SPRE SERVICII	54
2. PROTOCOALE DE REȚEA	57
2.1. MODELUL DE REFERINȚĂ TCP/IP	57
2.2. PROTOCOLUL HTTP	58
3. PROGRAMAREA CLIENT - SIDE.....	62
3.1. ELEMENTE AVANSATE DE HTML	62
3.1.1. <i>Tabele</i>	62
3.1.2. <i>Frames (cadre) în HTML</i>	65
3.1.3. <i>Formulare</i>	68
3.2. XML	72
3.3. CSS - CASCADING STYLE SHEETS (FOI DE STIL IN CASCADA)	73
4. PROGRAMAREA SERVER – SIDE CU ASP.NET.....	76
4.1. SERVERUL IIS.....	76
4.2. CARACTERISTICI ALE ASP SI ASP .NET	76
4.3. CREAREA DE APlicații WEB FOLOSIND ASP.NET.....	78
4.3.1. <i>Configurarea masinii de lucru pentru proiecte ASP.NET</i>	78
4.3.2. <i>Proiectele ASP.NET in Visual Studio .NET cu C#</i>	78
4.3.3. <i>Formulare în ASP.NET</i>	79
4.3.4. <i>Controale în ASP.NET</i>	80
4.3.5. <i>Pastrarea informatiilor</i>	81
4.3.6. <i>Navigarea intre Forms cu pastrarea informatiilor</i>	81
4.3.7. <i>Securitatea în ASP.NET</i>	82
4.4. ADO.NET	83
4.4.1. <i>Obiectele ADO.Net</i>	85
4.4.2. <i>Configurare, mentenanta</i>	86
5. REFERINTE	88

1. Arhitecturi Software

Dezvoltarea de aplicatii / programe software implica mai multe etape, programarea (scrierea de cod propriu-zisa) fiind doar una dintre ele si poate sa inseamne doar aproximativ 25% din timpul total de dezvoltare. Etapele sunt:

- Analiza cerintelor
- Proiectarea de nivel inalt (arhitectura)
- Proiectarea componentelor
- Implementarea
- Testarea si validarea
- Punerea in functiune
- Intretinerea

Proiectarea la nivel inalt, sau stabilirea arhitecturii are o importanta deosebita in acest proces, deoarece alegerile facute aici determina derularea viitoarelor etape, implicit limitele intre care se va programa efectiv aplicatia. Arhitectura este un model scris al unui sistem, care poate fi utilizat la construirea sistemului propriu-zis, si este formata din:

- Definirea scopului sistemului.
- Modelul de date.
- Diagrame de flux de date.
- Fluxul interfetei cu utilizatorul.

Există mai multe modele arhitecturale, dintre care putem aminti:

- Modelul client-server
- Multi-strat

Aplicatii orientate pe servicii

1.1. Modelul client- server

Termenul client-server este folosit prima data in 1980, legat de calculatoarele dîntr-o rețea. Arhitectura software client-server este o infrastructură modulară, bazată pe mesaje. Un client este definit ca un solicitant de servicii, iar serverul ca un prestator de servicii. O masină (un calculator) poate fi in același timp atât client cât și server în funcție de aplicațiile rulate.

Precursorale ale modelului client-server au fost arhitecturile de tip **mainframe** si **file sharing**. Propozitia precedenta nu e prea corecta, pentru ca se refera strict la modul in care erau utilize calculatoarele in trecut.

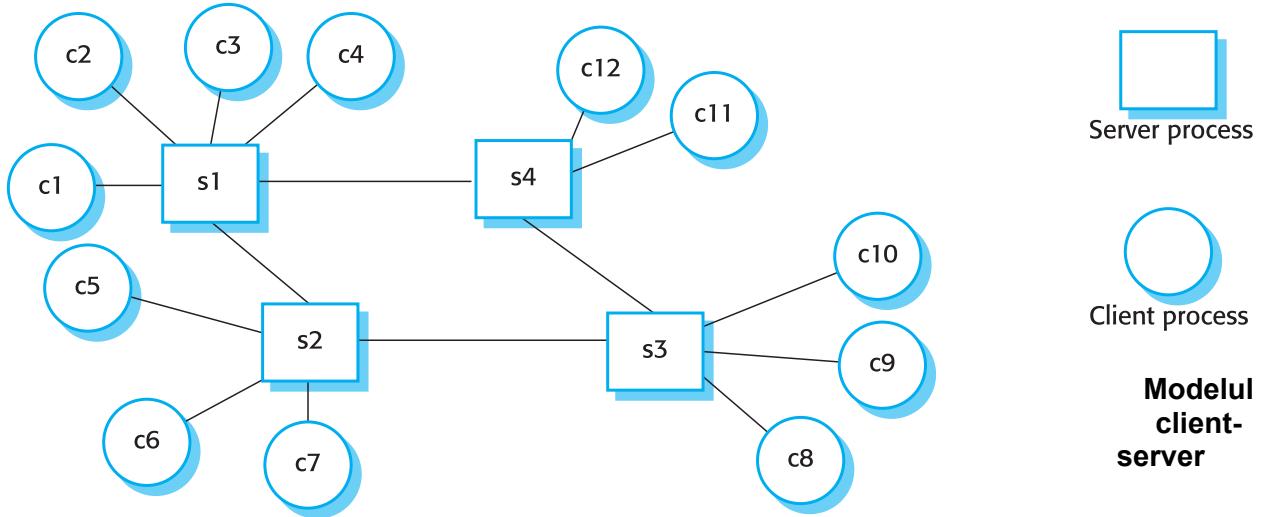
Urmatoarele paragrafe nu prea au legatura cu modelul arhitectural client-server, ci tot cu modul in care erau utilize calculatoarele. In plus, sunt si niste erori in exprimare, cum ar fi „Comunicarea între client și server se realizează prin intermediul interogarilor SQL sau a RPC (Remote Procedure Calls)”. Una este SQL si cu totul altceva este RPC – nu prea au legatura.

În arhitectura mainframe aplicațiile rulau pe un computer principal, iar utilizatorii interacționau cu acesta printr-un terminal (blind terminal) care trimitea informațiile tastate. Limitările acestui tip de arhitectură constau în faptul că nu asigurau o interfață grafică cu utilizatorul (GUI) și nu puteau accesa baze de date distribuite in diferite locații.

Rețelele de calculatoare inițiale se bazau pe arhitectura de tip partajare de fisiere (file-sharing), în care serverul download-a fisiere (dîntr-o locație partajată) pe calculatorul gazda. Aici se rulau aplicațiile (datele si procesarea lor). Viteza de lucru era scazută, iar numarul maxim de utilizatori, care accesa simultan resursele partajate, era mic.

Ca urmare a limitărilor arhitecturii de tip file-sharing, a aparut modelul client – server, în care serverul de fisiere a fost inlocuit de un server de baze de date. Prin utilizarea unui sistem de gestiune a bazelor de date (SGBD) se raspunde in mod direct cererilor utilizatorului. De asemenea se reduce traficul in retea prin returnarea unui raspuns la o interogare, față de returnarea unui întreg fisier. Apar si interfețele grafice (GUI) pentru

accesul la baza de date partajata (interfețe numite și front-end). Comunicarea între client și server se realizează prin intermediul interogarilor SQL sau a RPC (Remote Procedure Calls).



Mecanismul RPC a aparut cu mai bine de doua decenii in urma in lumea Unix si este folosit in construcția de aplicații distribuite pe sisteme eterogene, avand la baza tehnologiile Internet. O aplicatie RPC va consta dintr-un client si un server, serverul fiind localizat pe mașina care execută procedura. Aplicația client comunica prin rețea – via TCP/IP – cu procedura de pe calculatorul aflat la distanță, transmînd argumentele și recepcionând rezultatele. Clientul și serverul se execută ca două procese separate care pot rula pe calculatoare diferite din rețea.

In loc de ultimele 4 paragrafe, as insista mai mult pe niste modele client-server foarte cunoscute, cum ar fi:

- Windows Live Messenger. In acest caz, clientul este programul pe care il avem instalat pe calculatorul personal cei care dorim să comunicam prin acest serviciu. Server-ul (in acest caz programul server este instalat de fapt mai multe masini) este programul „central” care:
 - Are baza de date a utilizatorilor, adica a celor care au conturi de Windows Live.
 - Primeste mesaje de la clienti si le „ruteaza”, adica le trimite mai departe catre clientul / clientii (in cazul unei conferinte) destinatari.
 - Tine evidenta sesiunilor clientilor, adica „stie” tot timpul cand cineva este logged in, cu Live Messenger pornit, respectiv candiese.
 - Etc.

Solutiile de email, cum ar fi de exemplu serverul Exchange + Outlook.

1.2. Modelul multi-strat

Arhitecturile multi-strat sunt aceleia in care fiecare strat logic:

- Are un scop precis.
- Isi vede numai de propriile responsabilitati.
- Stie exact modul de comunicare / interactiune cu straturile adiacente.

In astfel de arhitecturi, este usor de separat anumite functionalitati care nu ar trebui să depindă unele de altele. De exemplu, interfata cu utilizatorul, sau mai bine spus clasele / paginile care formează aceasta interfata, nu trebuie să „stie” cum anume se realizează accesul la baza de date. Cu alte cuvinte, in clasele / paginile in care se afisează informații,

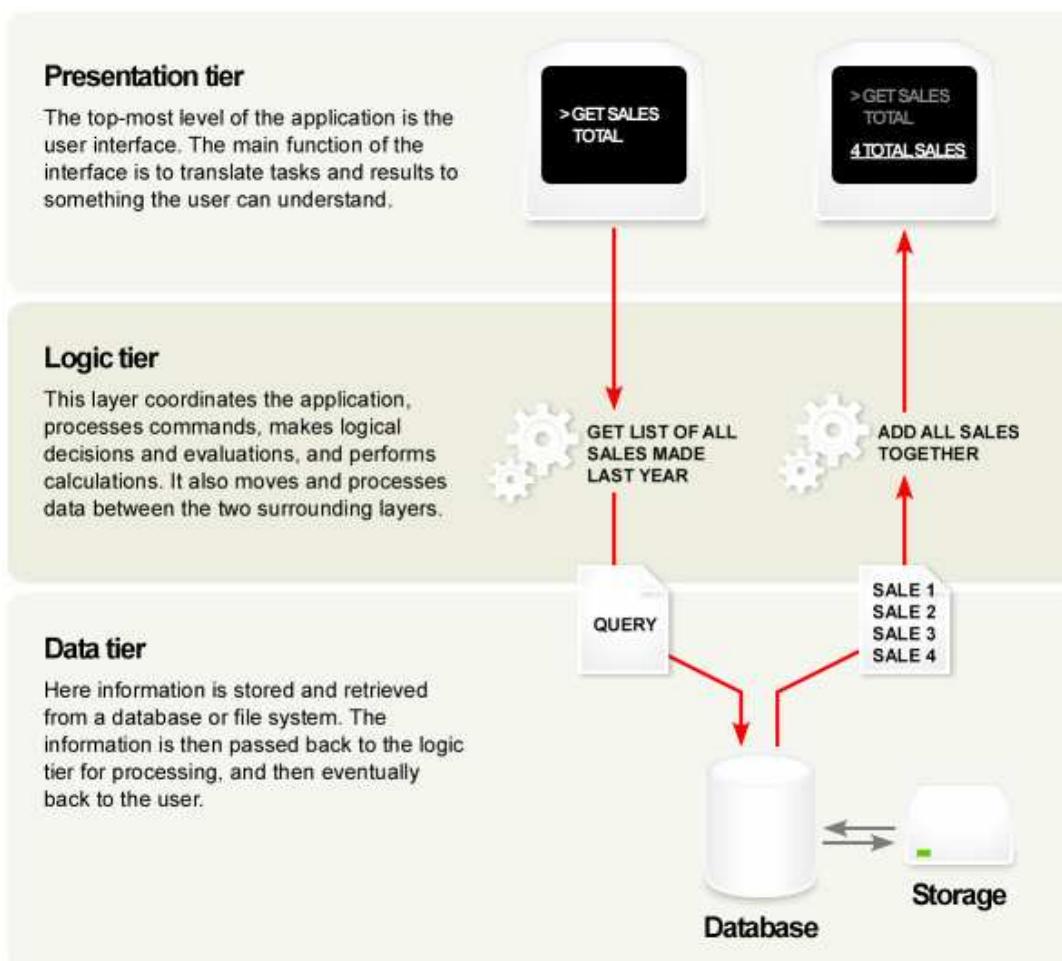
nu este nevoie să avem cod care lucreaza cu baza de date. Astfel, se realizeaza o separare logica benefica pentru intelegera mai usoara si intretinerea mai eficienta a software-ului.

De exemplu, putem avea arhitecturi multi-strat pe 2 nivele (two tier architecture), in care să spunem ca interfața cu utilizatorul este situată pe calculatorul utilizatorului sub forma unei aplicatii Windows (deci instalata pe calculator), iar sistemul de gestiune a bazelor de date pe un calculator mai puternic (server) care deserveste mai multi clienti

O alta arhitectura pe 2 nivele poate fi o aplicatie web in care interfata propriu-zisa (paginile, codul de afisare a informatiilor si de captare a datelor introduse de utilizator) este instalata pe un server, iar baza de date pe un alt server.

În arhitectura multi-strat pe 3 nivele (three tier architecture) se introduce un nou strat intre interfață și SGBD. Interfața cu utilizatorul , procesarea, stocarea si accesarea datelor sunt module separate, care se pot situa pe platforme diferite. Acest model asigură o flexibilitate sporita, fiecare modul putand fi modificat independent de celelalte. Spre exemplu schimbarea sistemului de operare din Windows în Linux poate afecta doar modulul de interfață.

De obicei, interfața cu utilizatorul rulează pe un alt calculator și folosește o interfață grafica standard (GUI), procesarea logica poate consta din unul sau mai multe module care ruleaza pe un server de aplicatii, iar SGBD-ul ruleaza pe un server de baze de date sau mainframe. Stratul de mijloc poate fi format la randul lui din mai multe nivele, caz in care arhitectura este numită multi strat cu n nivele (n-tier architecture).



Cele 3 nivele sunt:

- Nivelul de prezentare (presentation tier) sau interfață cu utilizatorul: prezentarea rezultatelor sistemului către utilizator și preluarea de informații de la utilizator
- Nivelul logic de procesare (logic tier / business logic tier/ transaction tier) : funcționalitatea specifică aplicației.

Nivelul de date (data tier): Interacțiunea cu suportul datelor (baze de date, fisiere, servicii web, etc).

1.3. Aplicații orientate spre servicii

Pentru a discuta despre Arhitecturi Orientate pe Servicii (SOA), ar trebui să plecam mai intai de la definitia acestora. Din pacate nu exista o definitie universal acceptat pentru SOA. Ceea ce se poate spune in schimb despre acest mode este ca prin intermediul lui se incearca descrierea unor arhitecturi bazate pe servicii slab interconectate care suporta modelarea proceselor de business si a necesitatilor utilizatorilor. Resursele aflate într-o „reteaua” sunt expuse ca servicii independente care pot fi accesate fara cunoștiințe apriori despre ceea ce rezida in spatele lor: platforma, implementare, algoritmi, etc.

Aceste concepte generice sunt valabile fie ca se vorbeste despre business-uri, software sau alte tipuri de sisteme de tip consumator-producator.

Dupa cum mentionam anterior, arhitecturile de tip SOA nu impun utilizare unei anumite tehnologii [precum REST, RPC, DCOM, CORBA sau Servicii Web]. Conceptul cheie care permite acest lucru este faptul ca serviciile sunt independente, si pot fi apelate într-un mod standardizat pentru a-si indeplini sarcinile pentru care au fost proiectate si implementate, fara ca serviciul să fie nevoie să stie in prealabil nimic despre aplicatia care il invoca, sau ca aplicatia care invoca serviciul să aibă nevoie să inteleaga mecanismele interne prin care serviciul isi duce la bun sfarsit rolul pentru care a fost definit.

Putem astfel incerca să definim SOA ca fiind: o paradigma pentru organizarea si utilizarea capabilitatilor distribuite care pot să fie sub controlul unor entitati diferite. Aceasta paradaigma ofera o modalitate uniforma de descoperire si interactioanre.

Conceptul de baza al SOA este aceala de interconectare slaba, realizat prin constrangeri arhitecturale precum:

- Fiecare serviciu ofera un set de interfete simple si care ofera doar o semantica generala, si care este accesibila de catre orice producator sau consumator interesat.
- Intre interfetele diferitelor servicii se schimba doar mesaje care respecta o schema predefinita, dar extensibila, prin intermediul carora se trimit doar continut si nu descriere comportamentala a sistemului. Cu alte cuvinte mesajul trebuie să aibă o mare putere de descriere si nu una de „comanda”/instruire.

Pe langa conceptul de interconectare slaba, mai exista si urmatoarele concepte referitoare la principiile unei arhitecturi de tip SOA:

- Incapsularea Serviciului
- Contractul Serviciului – se refera la faptul ca serviciile sunt de acord să comunice pe baza unui agreement stipulat de catre unul sau mai multe servicii prin intermediul unor documente descriptive
- Abstractizarea Serviciului – un serviciu expune ceea ce face si nu cum o face
- Refolosirea Serviciului – in sine conceptul se serviciu implica reutilizare. Serviciile au fost concepte pentru a putea să fie reutilizate.
- Compunerea Serviciilor – colectii de servicii pot să fie coordonate si asambleate astfel in cat să formeza un nou serviciu, numit un serviciu compus.
- Autonomia Serviciului – un serviciu are control total asupra logicii pe care o incapsuleaza
- Descoperirea Serviciilor – serviciile sunt concepute astfel incat să isi descrie functionalitatea catre „exterior”, astfel incat să poata fi gasite si folosite prin intermediul unor mecanisme de descoperire.

Avand bazele puse pentru conceptul de SOA, in cele ce urmeaza se va incerca definirea Serviciilor Web pe baza acestor concepte. Serviciile Web nu sunt altceva decât o

modalitate prin care o arhitectura SOA este realizate, folosind un anumit set de protocoale si restrictii. In cazul concret al Serviciilor Web, urmatoarele doua restrictii sunt impuse:

- Interfetele serviciilor trebuie sa fie construite peste protocoale din Internet, precum HTTP, FTP sau SMTP.
- Exceptand atasamentele binare alea unor mesaje, toata comunicarea intre servicii trebuie sa fie facuta prin mesaje in format XML.

In momentul de fata exista doua mari clase de Servicii Web, si anume: SOAP si REST. In cele ce urmeaza vom discuta despre Servicii Web de tip SOAP si infrastructuri oferite de platforma .NET pentru implementarea/dezvoltarea, gazduire, rularea si mentenanta unor Servicii Web de tip SOAP. Decizia de a descrie mai indetaliat Serviciile Web de tip SOAP este aceea ca ele sunt recunoscute de piata ca un standard defacto, fapt vizibil si prin adoptia majoritara a acestora.

SOAP este un acronim pentru Simple Object Access Protocol. Clasa de Servicii Web de tip SOAP aduce urmatoarele constrangeri fata de cele mentionate mai sus:

- Cu exceptia atasamentelor binare, toate mesajele trebuie sa fie transportate prin intermediul SOAP.
- Descrierea unui Serviciu Web se va face folosind un limbaj de markup numit WSDL – Web Service Description Language.
- Existenta regisrelor UDDI (Universal Description Discovery and Integration) care au ca scop indexarea Serviciilor Web. UDDI-ul permite interogarea sa prin intermediul SOAP si faciliteaza accesul la WSDL-ul Serviciilor Web pe care le indexeaza.

SOAP nu face altceva decat sa defineasca un format pentru mesaje care pot fi trimise intre servicii web deasupra unui numar de protocoale internet. Cu alte cuvinte, SOAP se comporta ca un plic care muta continut dintr-un punct in altul.

Pentru exemplificare, mai jos este un mesaj SOAP:

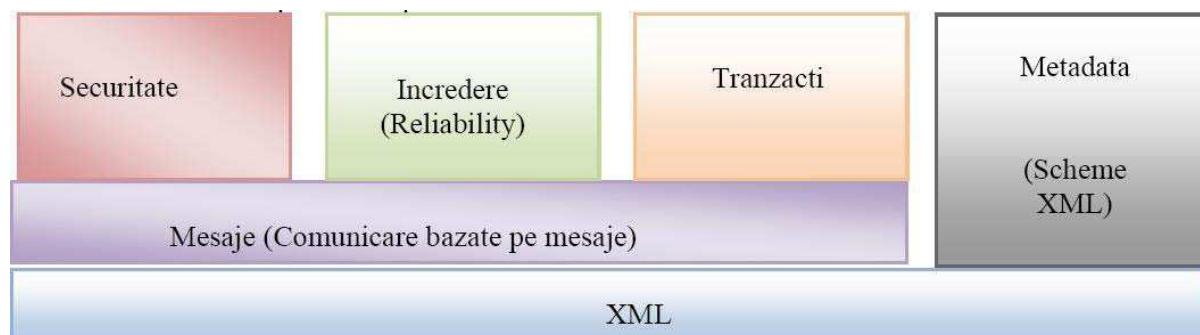
```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <obtineDetaliiCursant xmlns="http://www.example.com/WebService">
      <IDCursant>C010030</IDCursant >
    </obtineDetaliiCursant >
  </soap:Body>
</soap:Envelope>
```

Se poate observa in exemplul de mai sus structura unui mesaj de tip SOAP:

- Plicul: <soap:Envelope>
- Continutul acestui delimitat de: <soap:Body>

Odata cu dezvoltarea standardelor pentru Servicii Web de tip SOAP (de acum incolo numit simplu Servicii Web) si cu adoptia lor pe piata, s-a facut simtita nevoia unor standarde suplimentare pentru a putea exprima concepte precum securitatea, increderea si tranzactiile. O serie de firme din industria software au dezvoltat in consecinta o suita de standarde numite colectiv WS-* care trateaza aceste probleme. Telul acestor specificatii este acela de a oferi sabloane pentru functionalitati avansate ale Serviciilor Web, totodata pastrand simplitatea nativa a Serviciilor Web. Cea mai importanta trasatura a specificatiilor WS-* este posibilitatea lor de a fi compuse. Compunerea protocoalelor asociate acestor specificatii permite o dezvoltare incrementală a Serviciilor Web, pe masura ce functionalitatea lor o cere (securitate, incredere, atasamente, tranzactii, descoperire, etc.). Individual fiecare specificatie rezolva o problema izolata, dar ca un compus, ele rezolva probleme de functionalitate ridicata adesea intalnite in aplicatii distribuite.

Modelul acesta poate fi exprimat schematic astfel:



In cele ce urmeaza se vor prezenta succint produsele din platforma Microsoft care compun ecosistemul pentru construirea si mentenanta sistemelor distribuite implementate folosind Servicii Web.

Constructia de Servicii Web se poate realiza

- folosind Microsoft Visual Studio 2005 in colaborare cu platforma .NET 2.0, prin intermediul carora se pot implementa servicii web care sunt in conformitate cu standardul WS-Interoperability [WS-I]. Crearea serviciului este simpla, partea de contract fiind generata automat pe partea serviciului, iar partea de proxy fiind deasemenea generata automat pe partea de client. Suportul pentru descoperire este deasemenea integrat prin suportarea protocolului UDDI.
- folosind Web Service Enhancements 3.0 [WSE 3.0] in colaborare cu VS2005 si .NET 2.0, se pot realiza Servicii Web care sunt in conformitate cu ultimele specificatii ale WS-*. WSE 3.0 este oferit ca un pachet care se integreaza perfect cu suita de dezvoltare VS 2005. WSE 3.0 aduce suport in cadrul platformei .NET 2.0 pentru XML, SOAP, WSDL, WS-Security, WS-Trust, WS-SecureConversation, WS-Addressing si MTOM
- folosind platforma .NET 3.0 care aduce un set nou de API-uri printre care si Windows Communication Foundation, API care permite un model unificat de programare si rulare a Serviciilor Web folosind limbi managed, precum C#. WCF aduce suport integral pentru toate standardele recunoscute cat si cele mai populare adoptate de piata, numarand astfel urmatoarele:
 - o pentru trimitera mesajelor: XML, SOAP, WS-Addressing
 - o pentru descrierea interfelilor (asa numita MetaData): WSDL, WS-MetadataExchange, WS-Policy, WS-SecurityPolicy
 - o pentru securitate: WS-Trust, WS-Security, WS-SecureConversation
 - o pentru incredere: WS-ReliableMessaging

pentru tranzactii: WS-Coordination si WS-AtomicTransactions

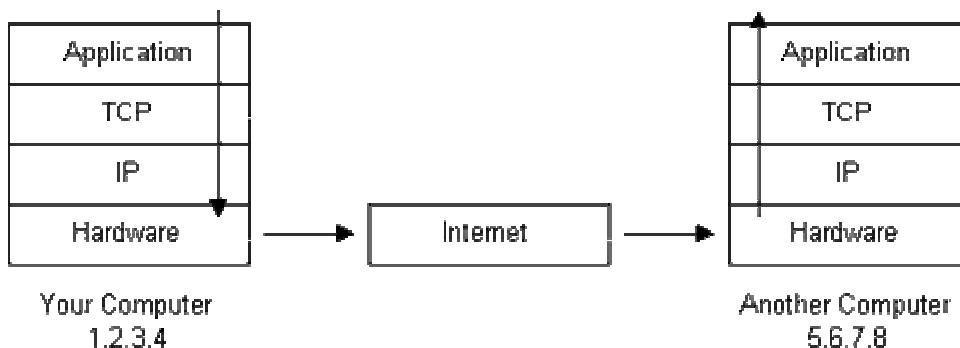
2. Protocole de rețea

2.1. Modelul de referință TCP/IP

Până în anii 60, comunicarea între computere se realiza greoi prin intermediul liniilor telefonice. În 1962 Paul Baran și Donald Davies propun – independent – ideea unui sistem de comunicare între rețele bazat pe comutarea de pachete – packet switching. Această arhitectură a fost implementată de ARPANET, o rețea de calculatoare sponsorizată de DoD (Department of Defence). Arhitectura a devenit cunoscută mai târziu sub denumirea de **modelul de referință TCP/IP**, după numele celor două protocole fundamentale utilizate. Protocolul TCP (Transfer Control Protocol) se referă la modalitatea de transmitere a informației, iar protocolul IP (Internet Protocol) se referă la modalitatea de adresare.

Dată fiind îngrijorarea DoD că o parte din gazde, routere și porturi de interconectare ar putea fi distruse la un moment dat, un obiectiv major a fost ca rețeaua să poată supraviețui pierderii echipamentelor din subrețea, fără a întrerupe conversațiile existente. Cu alte cuvinte, DoD dorea ca, atâtă timp cât funcționau mașina sursă și mașina destinație, conexiunile să rămână intace, chiar dacă o parte din mașini sau din liniile de transmisie erau brusc scoase din funcțiune. Mai mult, era nevoie de o arhitectură flexibilă, deoarece se aveau în vedere aplicații cu cerințe divergente, mergând de la transferul de fișiere până la transmiterea vorbirii în timp real.

Toate aceste cerințe au condus la alegerea unei rețele cu comutare de pachete bazată pe un nivel inter-rețea fără conexiuni. Acest nivel, numit nivelul internet (IP), este axul pe care se centrază întreaga arhitectură. Rolul său este de a permite gazdelor să emită pachete în orice rețea și face ca pachetele să circule independent până la destinație. Pachetele pot chiar să sosească într-o ordine diferită față de cea în care au fost trimise, caz în care rearanjarea cade în sarcina nivelurilor de mai sus (acele nivele ce urmează să proceseze pachetele primite).



Fiecare computer conectat la internet are o adresă de IP unică. Adresele IP sunt sub forma *nnn.nnn.nnn.nnn*, unde *nnn* trebuie să fie un număr între 0 și 255. Spre exemplu, la momentul redactării acestui document, www.microsoft.com are ip-ul 207.46.192.254. Fiecarui calculator îi este asociată o adresă de loopback 127.0.0.1, care înseamnă "acest calculator". 127.0.0.1 se mai numește și localhost. În anii '80 au fost conectate la Arpanet un număr foarte mare de rețele. Această situație a determinat, în 1984, crearea sistemului DNS (Domain Name System), prin intermediul căruia calculatoarele erau organizate în domenii, punându-se în corespondență adresele IP cu numele gazdelor.

Nivelul de transport sau TCP atașează pachetelor trimise un număr de port al serviciului destinatar și direcționează pachetele care ajung, către o anumită aplicație. Fiecare serviciu comunică pe un anumit port. Porturile sunt numere întregi care permit unui computer să facă diferență între comunicațiile internet ce au loc la un anumit moment dat. Numerele porturilor sunt cuprinse între 0 și 65535 și sunt împărțite în două clase: cele până la 1024 sunt rezervate pentru sistemul de operare, iar cele peste 1024 sunt disponibile dezvoltatorilor. Prin prezenta porturilor și deoarece aplicațiile care deservesc diferite servicii pe un sistem rulează pe porturi diferite, putem avea servicii care rulează concomitent pe

același calculator. Astfel, pe un sistem (server) pot rula concomitent aplicații server și pentru protocolul HTTP și pentru protocolul FTP. Un server poate deservi clienti atât ai serviciului sau HTTP cât și a celui FTP, deoarece poate deosebi pachetele adresate celor două servicii pe baza portului către care sunt trimise. Cele mai comune porturi sunt: FTP : 20/21, Telnet: 23, HTTP: 80, Microsoft SQL Server:1433.

Printre protocolele de nivel aplicatie, enumeram cîteva din cele mai cunoscute: HTTP, FTP, Telnet, SMTP, SSL, POP3, IMAP, SSH etc.

HTTP (Hyper Text Transfer Protocol) este un protocol client - server în care clientii (browserele web) trimit o cerere unui server web. După ce serverul returnează elementele cerute (imagini, text etc.) conexiunea dintre client și server este întreruptă. Acest lucru înseamnă că după ce elementele cerute sunt descarcate pe calculatorul utilizatorului, iar browserul descoperă că are nevoie de o alta resursă de pe server, se realizează o nouă cerere către acesta. Acest lucru înseamnă că HTTP nu este orientate pe conexiune (connection-oriented).

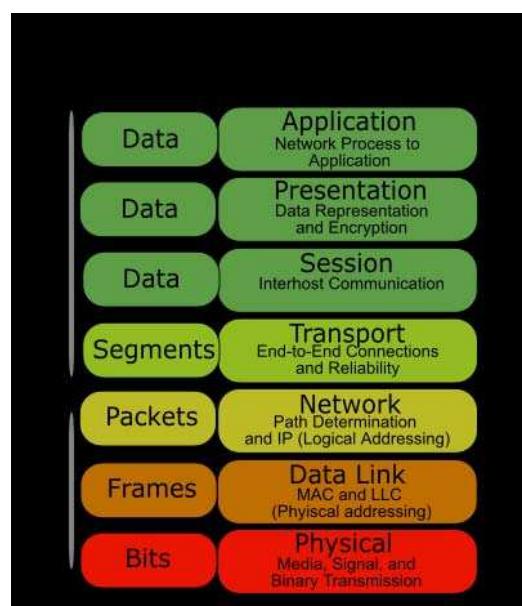
FTP (File Transfer Protocol) facilitează transferul de fișiere între două calculatoare din internet.

SMTP (Simple Mail Transfer Protocol) este un protocol folosit pentru schimbul de mesaje de poștă electronică.

Telnet este un protocol care permite unui utilizator să se conecteze de la un calculator la altul.

SSL (Secure Socket Layer) este un protocol folosit pentru transmiterea de date text criptate în internet.

Modelul TCP/IP este înrudit cu modelul de referință OSI (Open Systems Interconnection). Modelul OSI este format din 7 straturi asigurând astfel mai multă flexibilitate. Acest model nu este folosit însă în practică, rămânând mai mult un model teoretic. În figura de mai jos se poate vedea situația de comunicare propusă de modelul ISO/OSI cu cele 7 nivele ale sale. Nivelul cel mai de jos reprezintă nivelul fizic al rețelei, iar nivelul cel mai de sus reprezintă nivelul aplicație. În aceasta arhitectură multi-nivel, fiecare nivel comunica doar cu nivelele adiacente: la transmiterea de date, datele pleacă din nivelul aplicației, iar fiecare nivel comunica doar cu nivelul imediat de sub el; la primirea de date, fiecare nivel comunica doar cu nivelul imediat de deasupra lui.



2.2. Protocolul HTTP

Unul dintre cele mai importante și de succes servicii ale Internetului, **World-Wide Web**-ul – mai pe scurt **Web** sau spațiul **WWW**, a fost instituit la **CERN (Centre Européen**

pour la Recherche Nucléaire – Centrul European de Cercetări Nucleare de la Geneva) în anul 1989. Web – ul a apărut din necesitatea de a permite cercetătorilor răspândiți în lume să colaboreze utilizând colecții de rapoarte, planuri, desene, fotografii și alte tipuri de documente aflate într-o continuă dezvoltare.

Comunicația dintre clientul web (browserul) și serverul web are loc în felul următor: procesul server Web (spre exemplu Microsoft IIS – Internet Information Services) „ascultă” portul TCP 80 al mașinii gazde pentru a răspunde eventualelor cereri de conexiune venite de la clienți. După stabilirea conexiunii TCP, clientul care a inițiat conexiunea trimite o cerere (de exemplu un document HTML), la care serverul trimite un răspuns (documentul cerut sau un mesaj de eroare). După trimiterea răspunsului conexiunea dintre client și server se încheie. Cererile clientului și răspunsurile serverului Web sunt descrise de protocolul HTTP (Hyper Text Transfer Protocol).

Etapele care se parcurg între cerere și răspuns sunt:

1. Browserul determină URL – ul cerut de utilizator.
2. Browserul întreabă serverul local DNS adresa IP a mașinii pe care se află pagina cu URL-ul selectat.
3. Serverul DNS comunică browserului adresa IP cerută.
4. Browserul stabilește o conexiune TCP cu portul 80 al serverului cu IP – ul obținut anterior.
5. Utilizând conexiunea TCP, browserul trimite o comandă GET.
6. Serverul web răspunde trimițând un fișier HTML.
7. Serverul Web închide conexiunea TCP.
8. Browserul afișează textul din fișierul HTML trimis.
9. Browserul cere și afișează și celelalte elemente necesare în pagină (de exemplu imaginile).

Exemplu de cerere din partea clientului:

GET / HTTP/1.1
Host: www.live.com

Exemplu de răspuns al serverului:

HTTP/1.1 200 OK
Date: Wed, 15 Aug 2007 09:49:56 GMT
Server: Microsoft-IIS/6.0
P3P:CP="BUS CUR CONo FIN IVDo ONL OUR PHY SAMo TELo"
S:WEBA12
X-Powered-By: ASP.NET
X-AspNet-Version: 2.0.50727
Set-Cookie: mktstate=E=&P=&B=; domain=live.com; path=/
Set-Cookie: mkt1=norm=-ro; domain=live.com; path=/
Set-Cookie: mkt2=ui=ro-ro; domain=www.live.com; path=/
Set-Cookie: lastMarket=ro-ro; domain=.live.com; path=/
Set-Cookie: lastMktPath=ro/ro; domain=.live.com; path=/
Set-Cookie: frm=true; domain=.live.com; expires=Mon,
04-Oct-2021 19:00:00 GMT; path=/
Set-Cookie: frm=true; domain=.live.com; expires=Mon,
04-Oct-2021 19:00:00 GMT; path=/
Cache-Control: no-cache
Pragma: no-cache
Expires: -1
Content-Type: text/html; charset=utf-8
Content-Length: 5939

Metodele puse la dispoziție de protocolul HTTP sunt:

- HEAD – clientul cere numai antetul paginii

- GET - clientul cere serverului să trimită pagina specificată
- POST - trimit date pentru a fi procesate pe server
- PUT – înlocuiește conținutul paginii specificate cu datele trimise de client
- DELETE – solicită ștergerea paginii specificate
- TRACE – permite clientului să vadă cum sunt prelucrate mesajele
- OPTIONS – returnează metodele HTTP suportate de server
- CONNECT - convertește conexiunea de tip cerere într-un tunel TCP/IP , de obicei pentru a facilita comunicații securizate prin SSL (HTTPS).

Formularele HTML sunt folosite pentru ca utilizatorii să introducă date în câmpurile acestora și să le transmită serverului Web (utilizând browserul, o conexiune TCP și protocolul HTTP).

Cele mai răspândite servere Web sunt :

- IIS (Internet Information Server) de la Microsoft. Este inclus ca parte componentă a sistemelor de operare Microsoft (Windows NT Server, Windows 2000 Server și Windows Server 2003).

Apache (open source). Pentru a transmite date serverului se poate folosi metoda GET sau metoda POST. Metoda implicită este GET.

Dacă se utilizează GET, datele transmise serverului pot fi vizualizate în bara de adrese a browserului, ele urmând după URL-ul fișierului ce conține scriptul care le va prelucra. Datele introduse de utilizator vor fi sub forma de perechi nume=valoare. Caracterul ? semnifică faptul că se transmit date către server, iar perechile nume=valoare pot fi separate prin caracterul &. De exemplu pentru a căuta informații referitoare la ASP.NET în Live Search, se poate folosi url-ul: <http://search.live.com/results.aspx?q=ASP.NET>.

Metoda POST permite ca transmiterea datelor către server să se facă în flux, putându-se transmite cantități mari de date. De asemenea, datele trimise nu mai sunt afișate în bara de adrese a browserului, lucru util când se transmit informații confidențiale.

La nivelul serverului, datele primite de la browser (via HTTP) sunt prelucrate de un script, o aplicație sau chiar un framework:

- CGI (Common Gateway Interface). Reprezintă o serie de script-uri executate pe serverul web. Acestea pot fi scrise în orice limbaj de programare (interpretat sau compilat) cu respectarea următoarelor restricții: programul scrie datele la ieșirea standard și generează antete care permit serverului Web să interpreteze corect ieșirea scriptului, conform specificațiilor HTTP.
- PHP (initial Personal Home Page apoi PHP: Hypertext Preprocessor) și ASP (Active Server Pages). ASP (Active Server Pages) este o platformă de programare Web creată de Microsoft. Lucrează bine cu serverul IIS, dar există versiuni și pentru alte servere Web.

Desi difera din punct de vedere al sintaxei, atât PHP cât și ASP sunt interpretate, codul lor fiind stocat în fisiere externe cu extensia .php/.asp. De fapt, ASP nu oferă un limbaj nou, ci se bazează pe limbajele VBScript și JScript. Un fisier PHP/ASP poate fi combinat cu date de tip text, marcatori HTML și comenzi script. În momentul executiei, în urma cererii unui client Web, fisierul este procesat, fiecare script din cadrul lui este interpretat și rezultatul executiei este introdus înapoi în fisierul static HTML înainte ca rezultatul să fie trimis către client (browser web). Aceste modele completează destul de bine suportul dezvoltării aplicațiilor Web, însă aduc unele limitări: sunt lente deoarece la fiecare accesare fisierele sunt procesate și interpretate (în loc să fie compilate), nu sunt capabile să construiască controale reutilizabile care să incapsuleze funcționalități complexe pentru interacțiunea cu utilizatorul.

- Perl (Practical Extraction and Report Language) este un limbaj de programare complex, precursor al PHP
- JSP (JavaServer Pages) este o platformă de programare Web creată de Sun Microsystems și face parte din specificația Java 2 Enterprise Edition (J2EE).

JSP utilizează sintaxa XML și o serie de clase și funcții Java. Codul JSP este compilat de *servlet-uri* binare pentru o procesare rapidă.

Server-side scripting reprezintă „programarea” comportamentului serverului, iar client-side scripting se referă la „programarea” comportamentului browserului de pe client. Programarea la nivel de client se realizează în special prin JavaScript.

3. Programarea client - side

Unul din primele elemente fundamentale ale WWW (World Wide Web) este HTML (Hypertext Markup Language), care descrie formatul primar in care documentele sunt distribuite si vazute pe Web. Multe din trasaturile lui, cum ar fi independenta fata de platforma, structurarea formatarii si legaturile hipertext, fac din el un foarte bun format pentru documentele Internet si Web.

Standardul oficial HTML este stabilit de World Wide Web Consortium (W3C), care este afiliat la Internet Engineering Task Force (IETF). W3C a enuntat cateva versiuni ale specificatiei HTML, printre care si HTML 2.0, HTML 3.0,HTML 3.2, HTML 4.0 si, cel mai recent, HTML 4.01. In acelasi timp, autorii de browsere, cum ar fi Netscape si Microsoft, au dezvoltat adesea propriile "extensii" HTML in afara procesului standard si le-au incorporat in browserele lor. In unele cazuri, cum ar fi tagul Netscape , aceste extensii au devenit standarde *de facto* adoptate de autorii de browsere.

3.1. Elemente avansate de HTML

3.1.1. Tabele

Tabelele ne permit sa cream o retea dreptunghiulara de domenii, fiecare domeniu avand propriile optiuni pentru culoarea fondului, culoarea textului, alinierea textului etc. Pentru a insera un tabel se folosesc etichetele corespondente **<table>...</table>**. Un tabel este format din randuri. Pentru a insera un rand intr-un tabel se folosesc etichetele **<tr>...</tr>** ("table row "= rand de tabel).

Un rand este format din mai multe celule ce contin date. O celula de date se introduce cu eticheta **<td>..</td>**.

```
<html>
  <head>
    <title>tabel</title>
  </head>
  <body>
    <h1 align=center>Un tabel simplu format din 4 linii si 2 coloane</h1>
    <hr>
    <table>
      <tr>
        <td>c11</td>
        <td>c11</td>
      </tr>
      <tr>
        <td>c21</td>
        <td>c22</td>
      </tr>
      <tr>
        <td>c31</td>
        <td>c32</td>
      </tr>
      <tr>
        <td>c41</td>
        <td>c42</td>
      </tr>
    </table>
  </body>
</html>
```

In mod prestabilit, un tabel nu are chenar. Pentru a adauga un chenar unui tabel, se utilizeaza un atribut al etichetei `<table>` numit border. Acest atribut poate primi ca valoare orice numar intreg (inclusiv 0) si reprezinta grosimea in pixeli a chenarului tabelului. Daca atributul border nu este urmata de o valoare atunci tabelul va avea o grosime prestatibila egala cu 1 pixel, o valoare egala cu 0 a grosimii semnifica absenta chenarului. Cand atributul border are o valoare nenula chenarul unui tabel are un aspect tridimensional.

```
<table border="2">
  <tr>
    <td>c11</td>
    <td>c11</td>
  </tr>
</table>
```

Pentru a alinia un tabel intr-o pagina Web se utilizeaza atributul align al etichetei `<table>`, cu urmatoarele valori posibile: " left " (valoarea prestatibila), " center " si "right ". Alinierea este importanta pentru textul ce inconjoara tabelul. Astfel :

- daca tabelul este aliniat stanga (`<table align="left">`), atunci textul care urmeaza dupa punctul de inserare al tabelului va fi dispus in partea dreapta a tabelului.
- daca tabelul este aliniat dreapta (`<table align="right">`), atunci textul care urmeaza dupa punctul de inserare al tabelului va fi dispus in partea stanga a tabelului.
- daca tabelul este aliniat pe centru (`<table align="center">`), atunci textul care urmeaza dupa punctul de inserare al tabelului va fi afisat pe toata latimea paginii, imediat sub tabel.

Culoarea de fond se stabileste cu ajutorul atributului bgcolor, care poate fi atasat intregului tabel prin eticheta `<table>`, unei linii prin eticheta `<tr>` sau celule de date prin eticheta `<td>`. Valorile pe care le poate primi bgcolor sunt cele cunoscute pentru o culoare. Daca in tabel sunt definite mai multe atribute bgcolor, atunci prioritatea este urmatoarea: `<td>`, `<tr>`, `<table>` (cu prioritate cea mai mica).

```
<html>
  <head>
    <title>tabel</title>
  </head>
  <body>
    <h1 align=center>Un tabel simplu colorat</h1>
    <hr>
    <table border="3" bgcolor="green">
      <tr>
        <td>verde 11</td>
        <td bgcolor="red">rosu 11</td>
      </tr>
      <tr bgcolor="blue">
        <td>albastru 21</td>
        <td bgcolor="yellow">galben 22</td>
      </tr>
      <tr bgcolor="cyan">
        <td>c31</td>
        <td>c32</td>
      </tr>
      <tr>
        <td>c41</td>
        <td bgcolor="white">c42</td>
      </tr>
    </table>
```

```
</body>
</html>
```

Culoarea textului din fiecare celula se poate stabili cu ajutorul expresiei: `...`.

Distanța dintre două celule vecine se definește cu ajutorul atributului `cellspacing` al etichetei `<table>`. Valorile acestui atribut pot fi numere întregi pozitive, inclusiv 0, și reprezintă distanța în pixeli dintre două celule vecine. Valoarea predefinită a atributului `cellspacing` este 2.

Distanța dintre marginea unei celule și continutul ei poate fi definită cu ajutorul atributului `cellpadding` al etichetei `<table>`. Valorile acestui atribut pot fi numere întregi pozitive, și reprezintă distanța în pixeli dintre celule și continutul ei. Valoarea predefinită a atributului `cellpadding` este 1.

Dimensiunile unui tabel - latimea și înaltimea - pot fi stabilite exact prin intermediul a două atribute, `width` și `height`, ale etichetei `<table>`. Valorile acestor atribute pot fi:

- numere întregi pozitive reprezentând latimea respectivă înaltețea în pixeli a tabelului;
- numere întregi între 1 și 100, urmate de semnul %, reprezentând fractiunea din latimea și înaltețea totală a paginii.

```
<html>
  <head>
    <title>tabel</title>
  </head>
  <body>
    <h1 align=center>Text</h1>
    <table width="100%" height="100%">
      <tr>
        <td align="center">
          <h2>Text centralat.</h2>
        </td>
      </tr>
    </table>
  </body>
</html>
```

Un tabel poate avea celule cu semnificația de cap de tabel. Aceste celule sunt introduse de eticheta `<th>` (de la "tabel header" = cap de tabel) în loc de `<td>`. Toate atribute care pot fi atașate etichetei `<td>` pot fi de asemenea atașate etichetei `<th>`. Continutul celulelor definite cu `<th>` este scris cu caractere aliniate și centralizate.

Alinierea pe orizontală a continutului unei celule se face cu ajutorul atributului `align` care poate lua valorile:

- left (la stanga);
- center (centralizat, valoarea predefinită);
- right (la dreapta);
- char (alinierea se face față de un caracter).

Alinierea pe verticală a continutului unei celule se face cu ajutorul atributului `valign` care poate lua valorile:

- baseline (la baza);
- bottom (jos);
- middle (la mijloc, valoarea predefinită);
- top (sus).

Aceste atribute pot fi atașate atât etichetei `<tr>` pentru a defini tuturor elementelor celulelor unui rand, cât și etichetelor `<td>` și `<th>` pentru a stabili alinierea textului într-o singura celula.

Un tabel trebuie privit ca o retea dreptunghiulara de celule.Cu ajutorul a doua atribute ale etichetelor <td> si <th>, o celula se poate extinde peste celule vecine.

Astfel:

- extinderea unei celule peste celulele din dreapta ei se face cu ajutorul atributului colspan, a carui valoare determina numarul de celule care se unifica.
- extinderea unei celule peste celulele dedesubt se face cu ajutorul atributului rowspan, a carui valoare determina numarul de celule care se unifica.

Sunt posibile extinderi simultane ale unei celule pe orizontala si pe verticala. In acest caz , in etichetele <td> si <th> vor fi prezente ambele atribute colspan si rowspan.

```
<html>
  <head>
    <title>table</title>
  </head>
  <body>
    <h1 align=center>Un tabel simplu cu chenar</h1>
    <hr>
    <table border="0">
      <tr>
        <td rowspan="3">c11</td>
        <br>c21<br>c31</td>
        <td>c12</td>
        <td colspan="2" rowspan="3">c13 , c4</td>
        <br>c23, c24<br>c33, c34</td>
      </tr>
      <tr>
        <td>c22</td>
      </tr>
      <tr>
        <td>c32</td>
      </tr>
      <tr>
        <td>c41</td>
        <td colspan="3" rowspan="2">c42, c43, c44</td>
      </tr>
    </table>
  </body>
</html>
```

Daca un tabel are celule vide, atunci aceste celule vor aparea in tabel fara un chenar de delimitare.

In scopul de a afisa un chenar pentru celule vide se utilizeaza in interiorul unei celule:

- dupa <td> se pune ;
- dupa <td> se pune
.

Caracterul (no breakable space) este de fapt caracterul spatiu.Un spatiu introdus prin intermediul acestui caracter nu va fi ignorat de browser.

3.1.2. Frames (cadre) in HTML

Ferestre sau (cadre) ne permit sa definim in fereastra browserului subferestre in care sa fie incarcate documente HTML diferite.Ferestrele sunt definite intr-un fisier HTML special , in care blocul <body>...</body> este inlocuit de blocul <frameset>...</frameset>. In interiorul acestui bloc, fiecare cadru este introdus prin eticheta <frame>.

Un atribut obligatoriu al etichetei <frame> este src, care primește ca valoare adresa URL a documentului HTML care va fi încărcat în acel frame. Definirea cadrelor se face prin împărțirea ferstrelor (și a subferstrelor) în linii și coloane:

- împărțirea unei ferestre într-un număr de subferestre de tip coloană se face cu ajutorul atributului cols al etichetei <frameset> ce descrie acea fereastră;
- împărțirea unei ferestre într-un număr de subferestre de tip linie se face cu ajutorul atributului rows al etichetei <frameset> ce descrie acea fereastră;
- valoarea atributelor cols și rows este o listă de elemente separate prin virgula, care descriu modul în care se face împărțirea.

Elementele listei pot fi:

- un număr întreg de pixeli;
- procente din dimensiunea ferestrei (număr între 1 și 99 terminat cu %);
- n* care înseamnă n parti din spațiul ramas;

Exemplu 1: cols=200,* ,50%,* înseamnă o împărțire în 4 subferestre, dintre care prima are 200 pixeli, a treia ocupă jumătate din spațiul total disponibil, iar a doua și a patra ocupă în mod egal restul de spațiu ramas disponibil.

Exemplu 2: cols=200,1*,50%,2* înseamnă o împărțire în 4 subferestre, dintre care prima are 200 pixeli, a treia ocupă jumătate din spațiul total disponibil iar a doua și a patra ocupă în mod egal restul de spațiu ramas disponibil, care se împarte în trei parti egale, a doua fereastră ocupând o parte, iar a patra ocupând 2 parti.

Observații:

- dacă mai multe elemente din lista sunt configurate cu *, atunci spațiul disponibil ramas pentru ele se va împărti în mod egal.
- o subfereastră poate fi un cadru (folosind <frame>) în care se va încărca un document HTML sau poate fi împărțita la randul ei la alte subferestre (folosind <frameset>).

```
<html>
  <head>
    <title>frames</title>
  </head>
  <frameset cols="*, *">
    <frame src="f1.html">
    <frame src="f2.html">
  </frameset>
</html>
```

În exemplul următor este creată o pagină Web cu trei cadre mixte. Pentru a o crea se procedează din aproape în aproape. Mai întâi, pagina este împărțită în două subferestre de tip coloană, după care a doua subfereastră este împărțită în două subferestre de tip linie.

```
<html>
  <head>
    <title>frames</title>
  </head>
  <frameset cols="20%, *">
    <frame src="p1.html">
    <frameset rows="*, *">
      <frame src="p2.html">
      <frame src="p3.html">
    </frameset>
  </frameset>
</html>
```

Pentru a stabili culoarea chenarului unui cadru se utilizează atributul bordercolor. Acest atribut primește ca valoare un nume de culoare sau o culoare definită în conformitate

cu modelul de culoare RGB. Atributul bordercolor poate fi atasat atat etichetei <frameset> pentru a stabili culoarea tuturor chenarelor cadrelor incluse, cat si etichetei <frame> pentru a stabili culoarea chenarului pentru un cadru individual.

Atributul border al etichetei <frameset> permite configurarea latimii chenarelor tuturor cadrelor la un numar dorit de pixeli. Valoarea prestatibila a atributului border este de 5 pixeli. O valoare de 0 pixeli va defini un cadru fara chenar.

```
<html>
<head><title>frames</title></head>
<frameset cols="20%,*" bordercolor="green" border="15">
<frame src="p1.html">
<frameset rows="*, *">
<frame src="p2.html" bordercolor="blue"> <frame src="p3.html">
</frameset>
</html>
```

Pentru a obtine cadre fara chenar se utilizeaza border="0". In mod prestatibil, chenarul unui cadru este afisat si are aspect tridimensional. Afisarea chenarului unui cadru se poate dezactivata daca se utilizeaza atributul frameborder cu valoare "no". Acest atribut poate fi atasat atat etichetei <frameset> (dezactivarea fiind valabila pentru toate cadrele incluse) cat si etichetei <frame> (dezactivarea fiind valabila numai pentru un singur cadru). Valorile posibile ale atributului frameborder sunt: "yes" - echivalent cu 1; "no" - echivalent cu 0;

```
<html>
<head><title>frames</title></head>
<frameset cols="20%,*" border="0">
<frame src="p1.html">
<frameset rows="*, *">
<frame src="p2.html">
<frame src="p3.html">
</frameset>
</html>
```

Atributul scrolling al etichetei <frame> este utilizat pentru a adauga unui cadru o bara de derulare care permite navigarea in interiorul documentului afisat in interiorul cadrului. Valorile posibile sunt:

- yes - barele de derulare sunt adaugate intotdeauna;
- no - barele de derulare nu sunt utilizabile;
- auto - barele de derulare sunt vizibile atunci cand este necesar

```
<html>
<head><title>frames</title></head>
<frameset cols="*, *, *">
<frame src="p.html" scrolling="yes" noresize>
<frame src="p.html" scrolling="no" noresize>
<frame src="p.html" scrolling="auto" noresize>
</frameset>
</html>
```

Atributul noresize al etichetei <frame> (fara nici o valoare suplimentara) daca este prezent, inhiba posibilitatea prestatibila a utilizatorului de a redimensiona cadrul cu ajutorul mouse-ului.

Un cadru intern este specificat prin intermediul blocului <iframe>...</iframe>. Un cadru intern se insereaza intr-o pagina Web in mod asemanator cu o imagine sau in modul in care se specifica marcapajul <frame>, asa cum rezulta din urmatorul exemplu: <iframe src="iframes_ex.html" height=40% width=50%> </iframe>

In acest caz fereastra de cadru intern care are 40% din inaltimea si 50% din latimea paginii curente.

Atributele acceptate de eticheta <iframe> sunt în parte preluate de la etichetele <frame> și <frameset>, cum ar fi: src, border, frameborder, bordercolor, marginheight, marginwidth, scrolling, name, noresize; sau de la eticheta vspace, hspace, align, width, height;

In mod prestatabil, la efectuarea unui clic pe o legatura noua pagina (catre care indica legatura) o inlocuieste pe cea curenta in aceeasi fereastra (acelasi cadru). Acest comportament se poate schimba in doua moduri:

- prin plasarea in blocul <head>...</head> a unui element <base> care precizeaza, prin atributul target numele ferestrei (cadrului) in care se vor incarca toate paginile noi referite de legaturile din pagina curenta conform sintaxei: <base target="nume_fereastra/frame_de_baza">
- prin plasarea in eticheta <a> a atributului target, care precizeaza numele ferestrei (cadrului) in care se va incarca pagina nou referita de legatura, conform sintaxei: ...

Daca este prezent atat un atribut target in <base> cat si un atribut target in <a>, atunci cele precizate de atributul target din <a> sunt prioritare.

Numele unui cadru este stabilit prin atributul name al etichetei <frame> conform sintaxei: <frame name="nume_frame">

In exemplul urmator este prezentata o pagina Web cu doua cadre. Toate legaturile din cadrul 1 incarca paginile in cadrul 2.

```
<html>
<head><title>frames</title></head>
<frameset cols="20%, *">
<frame src="left.html" name="left">
<frame src="p.html" name="main">
</frameset>
</html>
<html>
<head><title>frame_left</title> </head>
<body>
<a href="p1.html" target="main">
Fisierul1</a><br> <a href="p2.html" target="main">
Fisierul2</a><br> <a href="p3.html" target="main">
Fisierul3</a><br><br> <a href="p1.html" target="_blank">
Fis1 într-o fereastra nouă</a><br><br> <a href="p1.html" target="_self">
Fis1 în fereastra curentă</a><br><br> <a href="p.html" target="main">
Home</a><br>
</body>
</html>
```

Atributul target al etichetei <frame> accepta anumite valori :

- "_self" (incarcarea noii pagini are loc in cadrul curent);
- "_blank" (incarcarea noii pagini are loc intr-o fereastra noua anonima);
- "_parent" (incarcarea noii pagini are loc in cadrul parinte al cadrului curent daca acesta exista, altfel are loc in fereastra browserului curent);
- "_top" (incarcarea noii pagini are loc in fereastra browserului ce contine cadrul curent);

3.1.3. Formulare

Un formular este un ansamblu de zone active alcătuit din butoane, casete de selectie, campuri de editare etc. Formularele asigura construirea unor pagini Web care permit utilizatorilor sa introduca informatii si sa le transmita serverului. Formularele pot varia de la o simpla caseta de text , pentru introducerea unui sir de caractere pe post de cheie de

cautare, pana la o structura complexa, cu multiple sectiuni, care ofera facilitati puternice de transmisie a datelor.

Un formular este definit intr-un bloc delimitat de etichetele corespondente **<form>** si **</form>**.

Exista doua atribute importante ale elementului **<form>**.

1. Atributul **action** precizeaza ce se va intampla cu datele formularului odata ce acestea ajung la destinatie. De regula , valoarea atributului **action** este adresa URL a unui script aflat pe un server WWW care primeste datele formularului , efectueaza o prelucrare a lor si expedieaza catre utilizator un raspuns.

```
<form action="http://www.yahoo.com/cgi-bin/nume_fis.cgi">
```

Script-urile pot fi scrise in limbajele Perl, C, PHP, Unix shell. etc

2. Atributul **method** precizeaza metoda utilizata de browser pentru expedierea datelor formularului. Sunt posibile urmatoarele valori:

- **get** (valoarea implicita). In acest caz , datele din formular sunt adaugate la adresa URL precizata de atributul **action**; (nu sunt permise cantitati mari de date - maxim 1 Kb)
- **post** In acest caz datele sunt expediate separat. Sunt permise cantitati mari de date (ordinul MB)

Majoritatea elementelor unui formular sunt definite cu ajutorul etichetei **<input>**. Pentru a preciza tipul elementului se foloseste atributul **type** al etichetei **<input>**. Pentru un camp de editare, acest atribut primeste valoarea "text". Alte atribute pentru un element **<input>** sunt:

- atributul **name** ,permite atasarea unui nume fiecarui element al formularului.
- atributul **value** ,care permite atribuirea unei valori initiale unui element al formularului.

Un buton de expediere al unui formular se introduce cu ajutorul etichetei **<input>**, in care atributul **type** este configurat la valoarea "submit". Acest element poate primi un nume prin atributul **name**. Pe buton apare scris "Submit Query" sau valoarea atributului **value**, daca aceasta valoare a fost stabilita.

```
<form method="post" action="mailto:youremail@youremail.com" >  
First:<input type="text" name="First" size="12" maxlength="12" />  
Last:<input type="text" name="Last" size="24" maxlength="24" />  
<input type="submit" value="Send Email" />  
</form>
```

Pentru elementul **<input>** de tipul camp de editare (**type = "text"**) , alte doua atribute pot fi utile:

- atributul **size** specifica latimea campului de editare depaseste aceasta latime, atunci se executa automat o derulare acestui camp;
- atributul **maxlength** specifica numarul maxim de caractere pe care le poate primi un camp de editare; caracterele tastate peste numarul maxim sunt ignoreate.

Daca se utilizeaza eticheta **<input>** avand atributul **type** configurat la valoarea "password" , atunci in formular se introduce un element asemanator cu un camp de editare obisnuit (introdus prin **type="text"**). Toate atributele unui camp de editare raman valabile. Singura deosebire consta in faptul ca acest camp de editare nu afiseaza caracterele in clar, ci numai caractere *, care ascund de privirile altui utilizator aflat in apropiere valoarea introdusa intr-un asemenea camp. La expedierea formularului insa, valoarea tastata intr-un camp de tip "password" se transmite in clar.

```

<html>
<head><title>forms</title></head>
<body><h1>Un formular cu un buton reset</h1>
<hr>
<form action="mailto:test@yahoo.com" method="post">
Nume:<input type="text" name="nume" value="Numele"><br>
Prenume:<input type="text" name="prenume" value="Prenumele"><br>
Password:<input type="password" name="parola" ><br>
<input type="reset" value="sterge"> <input type="submit" value="trimite"> </form></body>
</html>

```

Butoanele radio permit alegerea , la un moment dat , a unei singure variante din mai multe posibile. Butoanele radio se introduc prin eticheta `<input>` cu atributul type avand valoarea "radio".

Italian: `<input type="radio" name="food" />`
 Greek: `<input type="radio" name="food" />`
 Chinese: `<input type="radio" name="food" />`

O caseta de validare (checkbox) permite selectarea sau deselectarea unei optiuni. Pentru inserarea unei casete de validare se utilizeaza eticheta `<input>` cu atributul type configurat la valoarea "checkbox". Fiecare caseta poate avea un nume definit prin atributul name.fiecare caseta poate avea valoarea prestatibila "selectat" definita prin atributul checked.

```

<p>Please select every sport that you play.</p>
Soccer: <input type="checkbox" checked="yes" name="sports" value="soccer" />
<br />
Football: <input type="checkbox" name="sports" value="football" />
<br />
Baseball: <input type="checkbox" name="sports" value="baseball" />
<br />
Basketball: <input type="checkbox" checked="yes" name="sports" value="basketball" />

```

Intr-o pereche "name = value" a unui formular se poate folosi intregul continut al unui fisier pe post de valoare. Pentru aceasta se insereaza un element `<input>` intr-un formular , cu atributul type avand valoarea "file" (fisier). Atributele pentru un element de tip caseta de fisiere:

- atributul `name` permite atasarea unui nume
- atributul `value` primeste ca valoare adresa URL a fisierului care va fi expediat o data cu formularul. Aceasta valoare poate fi atribuita direct atributului `value`, se poate fi tastata intr-un camp de editare ce apare o data cu formularul sau poate fi selectata prin intermediul unei casete de tip File Upload sau Choose File care apare la apasarea butonului Browse... din formular;
- atributul `enctype` precizeaza metoda utilizata la criptarea fisierului de expediat.Valoarea acestui atribut este "multipart/form-data".

O lista de selectie permite utilizatorului sa aleaga unul sau mai multe elemente dintr-o lista finita. Lista de selectie este inclusa in formular cu ajutorul etichetelor corespondente `<select>si</select>`.

O lista de selectie poate avea urmatoarele attribute:

- atributul name, care ataseaza listei un nume (utilizat in perechile "name=value" expediat serverului);
- atributul size, care precizeaza (printr-un numar intreg pozitiv , valoarea prestabilita fiind 1) cate elemente din lista sunt vizibile la un moment dat pe ecran (celelalte devenind vizibile prin actionarea barei de derulare atasate automat listei).

Elementele unei liste de selectie sunt incluse in lista cu ajutorul etichetei `<option>`.

Doua atribute ale etichetei option se dovedesc utile:

- atributul value primeste ca valoare un text care va fi expediat serverului in perechea "name=value"; daca acest atribut lipseste , atunci catre server va fi expediat textul ce urmeaza dupa `<option>`;
- atributul selected (fara alte valori) permite selectarea prestabilita a unui element al listei.

```
<select>
<option>Bucuresti -- B</option>
<option>Craiova -- DJ</option>
<option selected="yes">lasi -- IS</option>
</select>
```

O lista de selectie ce permite selectii multiple se creeaza intocmai ca o lista de selectie obisnuita. In plus, eticheta `<select>` are un atribut multiple (fara alte valori). Cand formularul este expediat catre server pentru fiecare element selectat al listei care este se insereaza cate o pereche "name=value" unde name este numele listei.

```
<select multiple="yes" size="3" name="orase">
<option>Bucuresti -- B</option>
<option>Craiova -- DJ</option>
<option selected="yes">lasi -- IS</option>
</select>
```

Intr-un formular campuri de editare multilinie pot fi incluse cu ajutorul etichetei `<textarea>`. Eticheta are urmatoarele atribute:

- atributul cols, care specifica numarul de caractere afisate intr-o linie;
- atributul rows, care specifica numarul de linii afisate simultan;
- atributul name, care permite atasarea unui nume campului de editare multilinie;
- atributul wrap, (de la "word wrap"=trecerea cuvintelor pe randul urmator0, care determina comportamentul campului de editare fata de sfarsitul de linie.

Acest atribut poate primi urmatoarele valori:

- a) " off "; in acest caz:
 - intreruperea cuvintelor la marginea dreapta a editorului se produce numai cand doreste utilizatorul;
 - caracterul de sfarsit de linie este inclus in textul transmis serverului o data cu formularul;
- b) " hard "; in acest caz:
 - se produce intreruperea cuvintelor la marginea dreapta a editorului ;
 - caracterul de sfarsit de linie este inclus in textul transmis serverului o data cu formularul;
- c) " soft "; in acest caz:
 - se produce intreruperea cuvintelor la marginea dreapta a editorului ;
 - nu se include caracterul de sfarsit de linie in textul transmis serverului o data cu formularul;

```
<textarea cols="20" rows="5" wrap="hard">
```

As you can see many times word wrapping is often the desired look for your textareas. Since it makes everything nice and easy to read.

```
</textarea>
```

```
<textarea cols="20" rows="5" wrap="hard" readonly="yes">
```

As you can see many times word wrapping is often the desired look for your text areas. Since it makes everything nice and easy to read.

```
</textarea>
```

Intr-un formular pot fi afisate butoane. Cand utilizatorul apasa un buton, se lanseaza in executie o functie de tratare a acestui eveniment. Limbajul HTML nu permite scrierea unor astfel de functii (acest lucru este posibil daca se utilizeaza limbajele Javascript sau Java). Pentru a insera un buton intr-un formular, se utilizeaza eticheta `<input>` avand atributul type configurat la valoarea "button". Alte doua atribute ale elementului "button" sunt:

- atributul `name`, care permite atasarea unui nume butonului
- atributul `value`, care primeste ca valoare textul ce va fi afisat pe buton.

Toate elementele cuprinse intr-un formular pot avea un atribut `disabled` care permite dezactivarea respectivului element si un atribut `readonly` care interzice modificarea continutului acestor elemente.

3.2. XML

Xml (eXtensible Markup Language) este un limbaj care permite definirea documentelor ce contin informatii intr-un mod structurat. Informatia structurata dintr-un document xml are atat continut (cuvinte, imagini, etc.), cat si o indicatie despre rolul pe care il are acel continut (de exemplu continutul din sectiunea header are alt rol fata de cel din footer, care la randul lui e diferit de continutul dintr-un tabel al unei baze de date). Toate documentele au o anumita structura. Aceste structuri se construiesc utilizand marcatori. W3.org se ocupa de standardizarea XML-ului, si specificatiile complete ale acestui limbaj se afla la adresa <http://www.w3.org/TR/WD-xml>.

In HTML, atat multimea de tag-uri (marcatori), cat si semantica asociata fiecarui tag sunt predefinite. Spre exemplu `<H1>` va reprezenta intotdeauna un header de nivel 1, iar tagul `<documentHTML>` nu are nici un sens (nu are asociata nici o valoare semantica). Standardul XML nu specifica nici semantica si nici multimea de tag-uri ale limbajului. De fapt XML este un metalimbaj pentru definirea limbajelor bazate pe marcatori (markup language). Cu alte cuvinte XML permite definirea de tag-uri precum si relatia dintre acestea. Deoarece nu exista o multime de tag-uri predefinita, nu exista nici o semantica asociata cu acestea. Definirea semantica documentelor XML este o sarcina a aplicatiilor care folosesc documentelor respective sau a foilor de stiluri

Avantaje XML:

- XML descrie continutul si modul de afisare al continutului;
- informatia este mai usor de gasit si manipulat daca este in format XML;
- XML ofera facilitati multiple si imbunatatite de prezentare (de exemplu folosind browserele)

Structura unui document XML este arborescentă:

```
<book>
    <title>My First XML</title>
    <prod id="33-657" media="paper"></prod>
    <chapter>Introduction to XML
        <para>What is HTML</para>
        <para>What is XML</para>
    </chapter>
    <chapter>XML Syntax
        <para>Elements must have a closing tag</para>
        <para>Elements must be properly nested</para>
    </chapter>
</book>
```

<book> este elementul rădăcină (root), title, prod și chapter sunt frați (siblings) și sunt copii (child) lui book.

3.3. CSS - Cascading Style Sheets (foi de stil in cascada)

Stilurile pun la dispozitia creatorilor de site-uri noi posibilitati de personalizare a paginilor Web. Un stil reprezinta un mod de a scrie un bloc de text (adica anumite valori pentru font, marime culoare, aliniere, distante fata de margini etc). Exista doua modalitati de a defini un stil:

- sintaxa CSS (Cascading Style Sheets);
- sintaxa Javascript.

O foaie este construită din reguli de stil care spun unui browser cum să arate un document. Regulile de stil sunt formate după cum urmează: selector { property: value }

Declarațiile de stil multiple pentru un singur selector pot fi despărțite de punct și virgulă:

```
selector { property1: value1; property2: value2 }
```

Ca exemplu, următorul segment de cod definește proprietățile culoare (color) și mărime-font (font-size) pentru elementele H1 și H2:

```
<head>
<title>exemplu css</title>
<style>
h1 { font-size: x-large; color: red }
h2 { font-size: large; color: blue }
</style>
</head>
```

Orice element HTML este un posibil selector CSS1. Selectorul este elementul care este legat la un stil particular. De exemplu, selectorul în declarația CSS: P { text-indent: 3em }, este P.

Declarațiile CSS pot fi introduse într-o pagina HTML în 3 moduri: inline, intern și extern.

1) Declarația inline se realizează cu ajutorul tag-ului style. De exemplu:

```
<p style="background: blue; color: white;">A new background and font color with inline CSS</p>
```

La modul general declarația este de forma:

```
<htmltag style="cssproperty1: value; cssproperty2: value;"> </htmltag>
```

O eroare des întâlnită este folosirea „,” în interiorul declarației de mai sus:

```
<p style="background: url("yellow_rock.gif");">Greșit</p>
```

```
<p style="background: url(yellow_rock.gif);">Corect</p>
```

2) Declarația internă se realizează cu ajutorul tag-ului <style> ... </style>

```
<html>
<head>
<style type="text/css">
p {color: white; }
body {background-color: black; }
</style>
</head>
<body>
<p>White text on a black background!</p>
</body>
</html>
```

3) Declarația externă. Atunci când se folosește CSS este de preferat să se separe codul HTML de cosdul CSS. De aceea, declarațiile CSS se pun de obicei într-un fișier separat, cu extensia .css . Avantajele unei asemenea abordari sunt:

- separarea conținutului paginii de design, crescând lizibilitatea codului html
- modificarea ușoară a design-ului unui site prin modificarea într-un singur fișier css decât în mai multe pagini html
- reutilizarea ușoară a codului css

Declarația externă se realizează prin intermediul tag-ului <link></link>:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="test.css" />
</head>
<body>
  <h3> A White Header </h3>
  <p> This paragraph has a blue font. The background color of this page is gray
because we changed it with CSS! </p>
</body>
</html>
```

Fișierul test.css trebuie să se afle în același director cu pagina html.

În CSS se pot defini aşa numitele clase. Un simplu selector poate avea diferite clase, astfel permitându-i aceluiași element să aibă diferite stiluri. Toate tagurile din pagina HTML cu aceeași clasă vor fi formataate identic.

```
p.html { color: #191970 }
p.css { color: #4b0082 }
```

Exemplul de mai sus a creat două clase, css și html pentru folosirea cu tag-ul P al HTML. Caracteristica CLASS (clasă) este folosită în HTML pentru a indica clasa unui element, ca de exemplu,

<P CLASS = html>Doar o clasă este permisă pentru un selector. </P>
<P CLASS = css> De exemplu, p.html.css este invalid.</P>
<P CLASS = html>Acest rând are aceeași culoare cu primul. </P>

Clasele pot fi declarate fără un element asociat:

```
.note { font-size: small }
```

În acest caz, clasa note poate fi folosită cu orice element.

Selectorii ID sunt desemnați cu scopul de a defini elemente în mod individual și nu mai multe elemente o dată ca în cazul claselor. Un selector ID este desemnat folosind indicatorul "#" care precede un nume. De exemplu, un selector ID ar putea fi desemnat astfel:

```
#text3M { text-indent: 3em }
```

La acest selector se va face referință în HTML folosind caracteristica ID:

```
<P ID=text3M>Textul are în față un spațiu de mărimea a trei litere `m`</P>
```

O proprietate este desemnată unui selector pentru a-i manipula stilul. Exemplele de proprietăți includ color, margin, și font.

Pentru a descrie formataările repetitive într-o foaie de stil, gruparea selectorilor și a declarațiilor este permisă. De exemplu, toate titlurile dintr-un document ar putea primi declarații identice printr-o grupare:

```
H1, H2, H3, H4, H5, H6 {color: red; font-family: sans-serif }
```

Toți selectorii care sunt interpuși în cadrul altor selectori vor moșteni valorile proprietăților selectorilor în cadrul cărora sunt interpuși, dacă nu se specifică altfel. De exemplu, o culoare definită pentru BODY va fi aplicată și textului dintr-un paragraf.

Există și cazuri în care selectorul conținut nu moștenește valorile selectorului care îl conține, dar acestea ar trebui să se evidențieze în mod logic. De exemplu, proprietatea margin-top nu este moștenită; în mod intuitiv, un paragraf nu va avea aceeași margine de sus cum o va avea tot documentul (prin selectorul body).

O listă completă a selectorilor CSS puteți găsi la <http://www.w3schools.com/css/default.asp>

4. Programarea server – side cu ASP.NET

4.1. Serverul IIS

Microsoft Internet Information Services (IIS) este serverul de aplicatii web al Microsoft. El este o componenta a Windows XP: Add/Remove Programs -> Add/Remove Windows Components.

4.2. Caracteristici ale ASP si ASP .NET

ASP (Active Server Pages) reprezintă o tehnologie creată de Microsoft pentru a crea pagini web dinamice, ce are la bază stocarea și execuția scripturilor pe serverul Web.

Servelele Web care suportă tehnologia ASP sunt în număr de două:

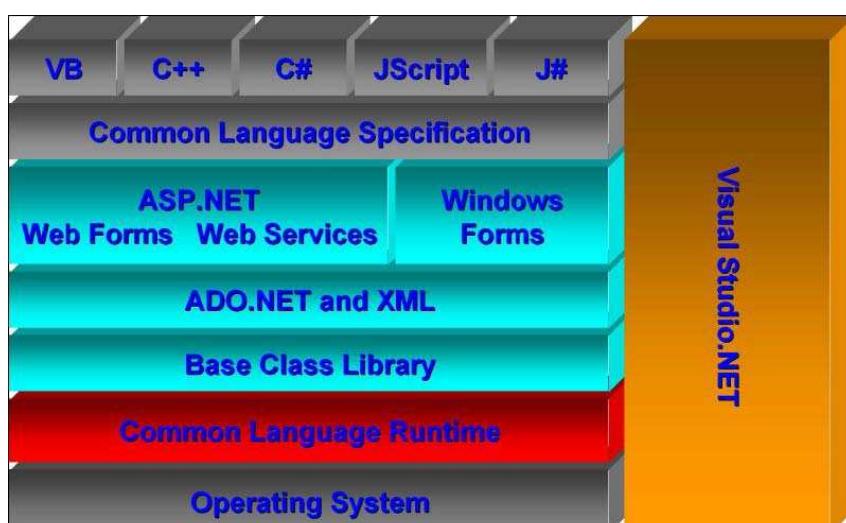
- **PWS** - Personal Web Server (Windows 98, Me) și
- **IIS** – Internet Information Server (Windows NT, 2000 și ulterior).

Principalele caracteristici ale tehnologiei ASP sunt:

- Permite accesarea și actualizarea usoară a bazelor de date;
- Viteză mare de execuție;
- Securitate ridicată, datorită faptului că scriptul nu poate fi vizualizat în browser;
- Generarea dinamică a răspunsurilor către clienți WEB, etc.

Microsoft .NET este o platformă de dezvoltare software, care are menirea de a oferi programatorilor un mediu în care sunt puse la dispozitie diverse servicii cum sunt: managementul fișelor de execuție, managementul duratei de viață a obiectelor (Garbage Collection), tratarea exceptiilor, etc; și care determină o viteză mult mai mare a dezvoltării aplicațiilor decât direct pe platforma Win32, adică direct pe sistemul de operare Windows.

Inovația adusă de Microsoft, constă în dezvoltarea unei platforme comune pentru mai multe limbi de programare, platformă care permite dezvoltarea unei aplicații în orice limbaj dorește programatorul. Acest lucru este posibil prin dezvoltarea unui interpretor – Common Language Runtime – care transformă codul de program dintr-un limbaj oarecare, compatibil cu platforma .NET, în limbajul **Microsoft Intermediate Language**. Codul rezultat în limbajul Microsoft Intermediate Language, se poate apoi compila pentru orice sistem de operare care are instalat .NET Framework. Paginile web dinamice create cu ajutorul ASP pot îngloba mai multe tipuri de tehnologii Web existente. De asemenea, scripturile din fisierele ASP pot fi și ele de mai multe tipuri :vbscript, javascript, active server)



Caracteristicile principale ale platformei .NET, sunt:

- Permite integrarea usoară a aplicațiilor Windows cu cele de WEB;

- Facilitează interoperabilitatea între aplicații, indiferent de platforma pentru care sunt create;
- Asigură serializarea/deserializarea obiectelor;
- Asigură un mediu stabil de programare obiectuală;
- Instalare simplificată, fără conflicte între versiuni;
- Execuția securizată a codurilor;
- Utilizează standardele existente pentru comunicare;
- Înlocuirea mediului script cu cel compilat, în cazul aplicațiilor ASP
- Codul de program al unei aplicații, scris într-un anumit limbaj de programare, poate fi integrat fără probleme în altă aplicație, scrisă în alt limbaj de programare, etc.;

Printre limbajele care suportă .NET, în primul rând trebuie amintite cele dezvoltate de Microsoft – VB .NET, C# .NET, Visual C++ .NET, cât și limbaje dezvoltate de alte firme: *Borland C#, Cobol, Eiffel, Perl, Python, Smalltalk, Pascal, Fortran*, etc.

Limbajele de programare dezvoltate de *Microsoft*, desi au toate la bază aceeași platformă, pot fi clasificate în funcție de gradul de complexitate:

- **VB .NET**, este cel mai accesibil, fiind o evoluție a mediului *Visual Basic 6.0*;
- **C# .NET**, este un limbaj „hibrid” între complexitatea limbajului C++ și accesibilitatea oferită de limbajul VB .NET;
- **VC ++ .NET** – este cel mai complex, fiind o evoluție a mediului VC++ 6.0.

ASP .NET reprezintă o evoluție a ASP bazată pe o nouă tehnologie dezvoltată de Microsoft, și anume platforma: .NET Framework. Tehnologia ASP .NET, aduce îmbunătățiri semnificative față de ASP, cele mai evidente fiind următoarele:

- Execuția este mai rapidă;
- Este independent de programul de navigare pe Internet;
- Codul aplicației este compilat și executat de server; acesta, în cazul ASP, este interpretat pe măsura parcurgerii scripturilor;
- Utilizează noțiunea de code behind, adică separă partea de prezentare de partea de execuție a unei aplicații WEB;
- Favorizează reutilizarea codului, ceea ce în cazul ASP simplu, era o problemă, singura „reutilizare” care se făcea în ASP, fiind aceea de copiere a codului;
- Serializarea/deserializarea cu usurință a obiectelor;
- Asigură interoperabilitatea între aplicații WEB, între aplicații WEB și alte categorii de aplicații;
- Securitate crescută;

Datorită platformei .NET, pot fi înglobate cu usurință în aplicațiile WEB toate componente care erau până acum caracteristice doar mediului Windows.

Microsoft ASP .NET este următoarea generație a tehnologiei de dezvoltare a aplicațiilor Web. Ea preia tot ce este mai bun de la Active Server Pages (ASP) la fel ca și serviciile bogate și facilitățile oferite de *Common Language Runtime* (CLR) și multe alte facilități noi. Rezultatul este o nouă modalitate de dezvoltare web rapidă, scalabilă și robustă care permite o mare flexibilitate cu foarte puține linii de cod scrise. **Web Forms** reprezintă partea centrală pe care se bazează ASP .NET. Acestea reprezintă elementele de interfață cu utilizatorul care dau aplicațiilor web aspectul și comportamentul dorit. Formularele Web sunt similare formularelor Windows prin faptul că oferă proprietăți, metode și evenimente controalelor plasate pe ele. Totusi, aceste elemente de interfață sunt afisate prin intermediul limbajului HTML, iar în cazul utilizării Microsoft Visual Studio .NET poate fi folosită interfață familiară de tip drag-and-drop pentru crearea aspectului formularelor Web. Formularele Web sunt constituite din două componente: partea vizuală (fisierul .ASPx) și codul din spatele formularului, care este stocat în fisiere separate.

4.3. Crearea de aplicații WEB folosind ASP.NET

Pentru a crea aplicații web cu ASP.NET avem nevoie de următoarele instrumente software:

Microsoft Internet Information Services (IIS) este serverul de aplicatii web al Microsoft. El este o componenta a Windows XP: Add/Remove Programs -> Add/Remove Windows Components.

Visual WebDeveloper 2005 Express Edition - este un mediu integrat de dezvoltare, care cuprinde inclusiv unelte de design al Form-urilor web, unelte de debug si deployment.

Colectia de **namespaces System.Web**. Acestea sunt parte integranta a .NET Framework si include clase predefinite care se ocupa de chestiuni specifice aplicatiilor web.

Controalele HTML si server sunt componente ale interfetei cu utilizatorul care sunt folosite pentru a afisa respectiv colecta informatii catre / dinspre utilizatori.

Limbajul de programare C#.

ADO.NET – sunt clase predefinite care se ocupa de managementul datelor – accesul la date in baze de date ODBC si Microsoft SQL Server.

4.3.1. Configurarea masinii de lucru pentru proiecte ASP.NET

Va trebui să aveți instalate IIS și Visual WebDeveloper 2005 Express Edition, care instaleaza automat si .NET Framework si inregistreaza ASP.NET in IIS.

Atentie: daca ati instalat IIS dupa .NET Framework, nu veti putea rula aplicatii ASP.NET. Pentru a face totusi asta, trebuie să rulati din linia de comanda fisierul **aspnet_regiis.exe** cu optiunea **-i**, din directorul C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322.

4.3.2. Proiectele ASP.NET in Visual Studio .NET cu C#

Un proiect ASP.NET poate fi creat cu ajutorul Visual WebDeveloper 2005 Express Edition: File -> New Web Site -> ASP.NET Web Site. Se alege la optiunea Language limbajul C#. Se creaza automat de catre IDE urmatoarele fisiere:

- Un Form care este de fapt un fisier **Default.aspx**, care poate fi vizualizat in doua moduri: Design si HTML. Acesta contine cod HTML si aici pot fi asezate prin design controalele care ruleaza la nivel de server.
- Fiecare fisier .aspx are asociat unul de tip **.cs**, unde este codul din spatele Form-ului, adica codul care este executat la nivel de server. Se poate ajunge la acel fisier cu click dreapta pe Form si View Code. Acest fisier are atasate automat cateva namespaces (folosind cuvantul cheie din limbajul C# **using**), printre care se pot observa si cele din familia System.Web.
- **Web.config** – acesta este un fisier XML in care se pot seta mai multe informatii de configurare a aplicatiei web. In general, el este folosit pentru securitate (autentificare), dar si pentru stocarea unor informatii gen constante care să poata fi regasite din codul executabil. Avantajele stocatii unor astfel de constante in web.config sunt: modificarile in acest fisier nu atrag dupa sine necesitatea de recompilare a aplicatiei, respectiv informatiile sunt într-un singur loc si pot fi modificate cu usurinta. (În fereastra solution explorer, click dreapta pe website1, add existing item și apoi web.config)

In general, se recomanda ca designul aplicatiilor (din punct de vedere al ingineriei software si nu vizual) să respecte anumite criterii. Astfel, daca ne gandim la modelul arhitectural pe mai multe nivele:

- User Interface Layer – Form-uri web, in care se fac afisari si de unde se preiau informatii de la utilizator.
- Business Layer – clase in care se efectueaza operatiile specifice aplicatiei.

- Data Access Layer – clase care se ocupa cu accesul la baza de date si trimitera informatiilor in nivelele superioare.

ASP.NET vine cu cateva controale predefinite, in plus fata de cele HTML, numite si controale server, numite asa deoarece tratarea evenimentelor la care acestea raspund se executa pe server. Visual Studio .NET 2005 are o interfata care permite manipularea facila a acestora din Toolbox.

ASP.NET 2.0 vine cu peste 50 de noi astfel de controale. Toolbox-ul este alcautuit din taburile Standard, Data, Validation, Navigation, Login, WebParts, HTML si General. Fiecare tab contine controale specifice.

In ASP.NET 2.0, majoritatea controalelor dispun de Smart Tasks (sau Common Tasks). Acestea, in functie de controlul in cauza, permit formatarea controalelor cu stiluri predefinite, atasarea de date, setarea modului de vizualizare etc.

Observati ca atunci cand introduceti controale intr-un Form web, in codul "din spatele acestuia", adica in fisierul .cs asociat, se produc niste modificari: se creaza un nou obiect, in functie de ce anume ati adaugat, ca instanta al uneia dintre clasele din spatiul de nume System.Web.UI.WebControls.

Exista controale folosite la afisare de date: Labels, TextBoxes; dar in mod curent se folosesc si unele pentru afisarea unor colectii de informatii, cum ar fi: ListBoxes, DropDownList si GridViews. Acestea din urma se folosesc foarte mult in aplicatii de management al informatiilor, cum ar fi de exemplu una de biblioteca, unde se poate dori afisarea tuturor cartilor dintr-un anumit domeniu, etc.

Controalele sunt caracterizate prin:

- Proprietati (ID-ul controlului, Text, Font, Activare etc.)
- Evenimente predefinite la care stiu sa raspunda (de exemplu, butoanele au Click, textbox-urile au TextChanged, etc.)

In momentul in care doriti sa tratiți un eveniment, sa spunem apasarea unui buton de catre utilizator, trebuie sa asociati un handler evenimentului predefinit Click al butonului respectiv. Un handler nu este altceva decat o functie, iar asocierea respectiva se poate face foarte usor din Visual Studio.

Pentru cazul anterior, este suficient dublu-click pe buton, pentru ca sa se creeze automat o metoda de tipul

```
protected void ButtonX_Click(object sender, EventArgs e){}
```

4.3.3. Formulare in ASP.NET

Pentru a prezenta informatii in navigatorul clientului folosim formularele Web ASP.NET care ofera o abstractizare in modelul de programare, un model orientat obiect si bazat pe evenimente. Acest mediu de lucru beneficiaza de toate facilitatile oferite de platforma .NET (siguranta tipurilor, mediu de executie controlat, mostenire) si reprezinta o inlocuire a clasicelor formulare HTML.

Componenta vizuala este reprezentata de un fisier cu extensia .aspx-actionand ca un container pentru HTML, text static si controale server care pot fi afisate in browser, iar logica aplicatiei este reprezentata de un fisier cu extensia .cs (pentru limbajul Visual C#) sau .vb (pentru Visual Basic.NET). Fisierele .aspx mai sunt referite ca pagini ASP.NET. Aceasta tehnica de separare a codului de partea de prezentare este numita "code-behind programming".

Formularele Web si ASP .NET au fost create pentru a depasi cateva dintre limitarile ASP. Principalele facilitati noi sunt redate in continuare:

- Separarea interfelei HTML de logica aplicatiei
- Un set bogat de controale pentru server ce detecteaza tipul browserului si genereaza limbaj HTML corespunzator acestuia
- Mai putin cod de scris din cauza modului in care sunt construite noile controale server
- Model de programare bazat pe evenimente

- Cod compilat si suport pentru mai multe limbaje de programare, fata de ASP care era interpretat ori ca VBScript ori ca Jscript
- Permite crearea de controale de catre terți care sa aducă noi funcționalități.

Logica aplicatiei reprezinta in fapt o clasa care extinde functionalitatea clasei System.Web.UI.Page. Aceasta clasa contine metode care trateaza diferite evenimente ce apar in timpul executiei formularului Web la server (de exemplu, daca metoda Page_Load este conectata la evenimentul Load al clasei de baza Page, atunci aceasta este apelata la fiecare acces al unui formular Web), proprietati (de exemplu, prin proprietatea IsPostBack putem afla daca o pagina Web este la primul acces sau la accesari ulterioare), atribute corespunzatoare unor controale din pagina WebForms si alte date membre necesare implementarii aplicatiei.

O pagina WebForms, la procesarea pe serverul Web, poate fi privita ca un program executabil pentru care iesirea standard o reprezinta browserul sau dispozitivul client. In acest model, pagina trece printr-o serie de stagii de procesare: initializare, procesare si eliberare. In ordinea aparitiei, acestea sunt:

- Init, eveniment care initializeaza pagina si in care proprietatile controalelor sunt actualizate. Aici este corect sa initializam controale care se adauga dinamic la pagina sau variabile necesare inainte de initializarea paginii;
- Load poate fi numit locul in care utilizatorul isi initializeaza codul. Evenimentul este generat de fiecare data cand pagina este incarcata dupa ce controalele au fost initializate la pasul anterior;
 - Tratarea evenimentelor utilizator, reprezinta stagiul in care sunt tratate evenimentele generate de client cum ar fi: schimbarea unui text intr-un control, apasarea unui buton etc. Trebuie retinut ca aceste evenimente nu sunt tratate intr-o anumita ordine pe server, iar tratarea lor are loc dupa aparitia unui eveniment Click care trimite formularul la server (a unui submit);
 - PreRender, eveniment care poate fi folosit pentru a face ultimile actualizari asupra paginii Web inainte ca aceasta sa fie generata la client;
 - Render, eveniment care genereaza la client reprezentarea HTML a paginii Web ASP.NET incarcata la server;
 - Unload este ultimul eveniment care se executa inainte ca pagina sa fie eliberata. Evenimentul este util de folosit atunci cand dorim sa efectuam ultimele operatii de eliberare a resurselor: inchiderea fisierelor, a conexiunilor la baza de date si eliberarea obiectelor din memorie.

4.3.4. Controale în ASP.NET

Exista doua tipuri de baza in care pot fi impartite controalele:

- HTML Controls, reprezinta elemente HTML care pot fi programate la nivelul serverului si expun un model obiectual restrictionat la capabilitatile elementelor HTML pe care le afiseaza;
- Web Controls, aduc facilitati superioare controalelor HTML incluzand controale mult mai complexe, cum ar fi controlul calendar, iar modelul obiect nu reflecta neaparat sintaxa HTML.

Controalele HTML sunt asemenea elementelor HTML folosite cu ajutorul Frontpage sau al oricărui alt editor HTML. Pot fi folosite si elementele standard HTML intr-un Formular Web, de exemplu pentru a crea o casetă de text: <input type="text" id=txtFirstName size=25>

Orice element poate fi însă marcat să ruleze ca si un control HTML atunci când formularul este procesat de server prin adăugarea textului "runat=server" în interiorul tagului.

<input type="text" id=txtFirstName size=25 runat=server>

În cazul folosirii Visual Studio .NET, adăugarea acestui text ce permite procesarea controlului de către server se poate face foarte simplu dând clic dreapta pe elementul HTML în Design View și selectând Run as Server Control din meniul contextual.

Controalele HTML permit de asemenea tratarea evenimentelor asociate cu tagul HTML (clic pe un buton, de exemplu) și manipularea în mod programatic a tagului prin codul din Formularul Web. În momentul în care controlul este afisat în browser, tagul este afisat exact așa cum a fost salvat ca și cum a fost salvat pe Formularul Web, mai puțin textul „runat=server”, ceea ce oferă un control foarte precis a ceea ce va fi trimis către browser.

ASP.NET definește un al doilea tip de controale - Web Controls. Numite și controale inteligente, ele pot fi caracterizate prin:

- oferă un bogat și consistent model obiectual de programare;
- detectează automat tipul navigatorului, iar afisarea la client va fi optimizată în funcție de capabilitățile acestuia;
- pentru unele controale se pot defini sabloane (template-uri) de afisare;
- posibilitatea de a controla generarea evenimentelor pe server;
- posibilitatea de a trimite evenimente unui container din interiorul acestuia (de exemplu, un control de tip buton în interiorul unui tabel);
- legarea la surse de date a tuturor proprietăților controalelor pentru a influența afisarea la execuție.

Sunt definite în spațiul de nume System.Web.UI.WebControls și mostenesc, direct sau indirect, clasa de bază WebControl.

4.3.5. Pastrarea informațiilor

Având în vedere că ne referim la aplicații ASP.NET, trebuie să tinem cont de faptul că fiecare Form se execută pe server și la fiecare încarcare a sa, obiectele pe care le conține acesta, și pe care noi dorim să le folosim, nu își pastrează valoarea, adică sunt setate pe null. Deci la navigarea într-un Form nou și încarcarea acestuia, în mod normal nu se rețin obiectele folosite în cel anterior.

Mai trebuie observat faptul că fiecare Form se reinictează (se execută Page_Load) la fiecare răspuns al serverului pentru un eveniment (cum ar fi apăsarea unui buton).

Se impune deci să pastrăm starea obiectelor pe care dorim să le folosim, chiar dacă nu parăsim Form-ul curent.

Există următoarele modalități de a pastra informații utile: Query Strings, Cookies, View State, Session State, Application State.

Session State: Variabilele din Session pot fi create în timpul execuției, iar acesta poate fi imaginat că un cos în care depunem obiecte pe care dorim să le pastrăm pe durata întregii execuții a aplicației de către utilizatorul curent. Astfel ar arăta crearea unui nou obiect în Session:

```
Session["Feedback"] = objFeedback;
Iar "luarea" lui din Session:
objFeedback = (Feedback)Session["Feedback"];
```

Se observă că s-a facut o conversie de tip. În Session, toate variabilele sunt de tip object, iar atribuirea se face unui obiect de tip Feedback (o clasa definită de utilizator).

4.3.6. Navigarea între Forms cu pastrarea informațiilor

Există mai multe posibilități de a naviga între Form-urile unei aplicații ASP.NET.

Controlul Hyperlink și metoda Response.Redirect() fac același lucru, diferența fiind că primul este un control care rezidează în .aspx, iar metoda se apelează din cod (.aspx.cs).

Metoda Server.Transfer(), face același lucru doar că astăzi se pot retine informații din Form-ul sursă și folosițe în cel de destinație. Dacă setăm parametrul preserveForm al metodei la valoarea true, atunci QueryString și ViewState din sursă vor fi vizibile în destinație. Totuși, pentru a le folosi, va trebui să setăm atributul EnableViewStateMac din directiva Page

(fisierul .aspx) pe valoarea false. Deoarece in mod normal informatia din ViewState este hash-uita, iar asa nu va mai fi, deci se va putea citi la nevoie

4.3.7. Securitatea în ASP.NET

Exista mai multe moduri de a securiza o aplicatie ASP.NET. In general, se poate face acest lucru atat din IIS Management Console (click dreapta pe un director virtual din IIS, Properties, Directory Security, Edit) – unde se pot alege mai multe tipuri de autentificare {Anonymous, Windows Integrated, Digest for Windows domain servers, Basic}.

Pe de alta parte, securizarea aplicatiei se poate face din ASP.NET. Pentru asta se foloseste fisierul de configurare web.config, dupa cum se vede mai jos. Forms poate fi inlocuit cu Windows sau Passport.

`<authentication mode="Forms">`

In primul rand, trebuie totusi sa vorbim de aplicatii publice care nu necesita nici un fel de autentificare. Trebuie sa fiti atenti, acesta este modul implicit in care este creata o noua aplicatie ASP.NET. Atunci cand se creeaza un proiect ASP.NET cu Visual Studio, se creaza o aplicatie web accesibila anonim.

In momentul in care in IIS se activeaza Anonymous Access pe un director virtual (adica pe o aplicatie web), atunci cand un utilizator acceseaza aplicatia, el ia automat contul IUSR_<numele_masinii>, care in mod normal se afla in grupul de utilizatori Guest pe orice masina. Acest cont trebuie sa aiba dreptul de a citi fisierele care sunt necesare aplicatiei. De exemplu, daca aplicatia foloseste la un moment dat Form-ul "Student.aspx", atunci IUSR_<numele_masinii> trebuie sa aiba dreptul de a citi fisierul de pe hard-disk corespunzator, sa spunem "c:\Inetpub\wwwroot\library\student.aspx".

Pentru o aplicatie securizata, avem mai multe posibilitati de autentificare, cele mai des intalnite fiind sintetizate in tabelul de pe slide. Implementarea politicii de securitate se poate face atat din IIS cat si din aplicatia ASP.NET.

Tipul aplicatiei	Modul de autentificare	Descriere
Aplicatie web publica pe Internet.	Anonim	Nu avem nevoie de securizare.
Aplicatie web pentru Intranet.	Windows Integrated	Acest mod autentifica utilizatorii folosind lista de useri de pe server (Domain Controller). Drepturile userilor in aplicatia web este dat de nivelul de privilegii al contului respectiv.
Aplicatie web disponibila pe Internet, dar cu acces privat.	Windows Integrated	Utilizatorii companiei pot accesa aplicatia din afara Intranetului, folosind conturi din lista serverului (Domain Controller).
Aplicatii web comerciale.	Forms Authentication	Aplicatii care au nevoie de informatii confidentiale si eventual in care sunt mai multe tipuri de utilizatori.

4.3.7.1.1 Windows Authentication

În acest mod de autentificare, aplicația ASP .NET are încorporate procedurile de autentificare, dar se bazează pe sistemul de operare Windows pentru autentificarea utilizatorului.

1. Utilizatorul solicită o pagină securizată de la aplicația Web.
2. Cererea ajunge la Serverul Web IIS care compară datele de autentificare ale utilizatorului cu cele ale aplicației (sau ale domeniului)
3. Dacă acestea două nu corespund, IIS refuză cererea utilizatorului.
4. Calculatorul clientului generează o fereastră de autentificare,
5. Clientul introduce datele de autentificare, după care retrimit cererea către IIS
6. IIS verifică datele de autentificare, și în cazul în care sunt corecte, direcționează cererea către aplicația Web.
7. Pagina securizată este returnată utilizatorului.

4.3.7.1.2 Forms-Based Authentication

Atunci când se utilizează autentificarea bazată pe formulare, IIS nu realizează autentificarea, deci este necesar ca în setările acestuia să fie permis accesul anonim.

1. În momentul în care un utilizator solicită o pagină securizată, IIS autentifică clientul ca fiind un utilizator anonim, după care trimit cererea către ASP.NET.
2. Aceasta verifică pe calculatorul clientului prezența unui anumit cookie1
3. Dacă cookie-ul nu este prezent sau este invalid, ASP .NET refuză cererea clientului și returnează o pagină de autentificare (Login.aspx).
4. Clientul completează informațiile cerute în pagina de autentificare și apoi trimit informațiile
5. Din nou, IIS autentifică clientul ca fiind un utilizator anonim și trimit cererea către ASP .NET
6. ASP .NET autentifică clientul pe baza informațiilor furnizate. De asemenea generează și un cookie.

Cookie reprezintă un mic fisier text ce păstrează diverse informații despre utilizatorul respectiv, informații folosite la următoarea vizită a să pe site-ul respectiv, la autentificare, sau în diverse alte scopuri.

7. Pagina securizată cerută și noul cookie sunt returnate clientului. Atât timp cât acest cookie rămâne valid, clientul poate solicita și vizualiza orice pagină securizată ce utilizează aceleasi informații de autentificare.

4.3.7.1.3 Passport Authentication

La utilizarea Serviciului Web Microsoft Passport, nici IIS nici aplicația Web nu se ocupă de autentificarea clientului. Atunci când utilizatorul cere o pagină securizată, cererea este trimisă mai întâi serverului IIS.

1. IIS autentifică clientul ca utilizator anonim și trimit cererea către ASP .NET.
2. Aceasta verifică prezența unui anumit cookie pe calculatorul clientului.
3. Dacă cookie-ul nu există, cererea este refuzată și clientul este direcționat către site-ul Web Passport.com pentru autentificare.
4. Site-ul Passport.com generează un formular de Login pe care îl trimit utilizatorului
5. Utilizatorul completează cu datele corespunzătoare și le trimit înapoi către Passport.com.
6. Dacă informațiile introduse se potrivesc cu cele din baza de date Passport.com, clientul este autentificat și primește un cookie corespunzător informațiilor introduse.

4.4. ADO.NET

ADO.NET este componenta din .NET Framework care se ocupa cu accesul la baze de date ; este standardizata in sensul ca se pot folosi aceleasi obiecte pentru accesarea

diferitelor tipuri de baze de date : Access, MS SQL Server, Oracle, etc. Sunt necesare la referinte doua namespaces : System.Data si System.Data.SqlClient pentru MS SQL sau System.Data.OleDb pentru Access.

Mecanismul accesarii bazelor de date in ADO.NET este urmatorul: un obiect Connection stabileste o conexiune intre aplicatie si baza de date. Aceasta conexiune poate fi accesata direct de un obiect Command sau de un obiect DataAdapter. Obiectul Command executa o comanda asupra bazei de date. Daca se returneaza valori multiple, se utilizeaza un obiect DataReader care va contine datele returnate. Aceste date pot fi procesate direct de la aplicatie. Alternativ, se poate utiliza un DataAdapter pentru a popula un obiect DataSet. Modificarile asupra bazei de date se pot efectua prin intermediul unui obiect Command sau unui obiect DataAdapter.

Connection reprezinta conexiunea curenta la baza de date.

Tipuri de conexiuni:

- SqlConnection - pentru conectarea la SQL Server 7 sau versiuni ulterioare
- OleDbConnection - conexiuni la diverse tipuri de baze de date
- ODBCConnection
- OracleConnection

Un obiect Connection contine toate informatiile necesare deschiderii unui canal de comunicatie cu baza de date in cadrul proprietatii ConnectionString. Sunt incorporate, de asemenea, metode pentru facilitarea tranzactiilor.

Command este reprezentat de doua clase: SqlCommand si OleDbCommand Utilizat pentru a efectua apeleuri de proceduri stocate sau de comenzi SQL asupra bazei de date sau pentru a returna tabele.

Metode:

- ExecuteNonQuery - executa comenzi care nu returneaza inregistrari - INSERT, UPDATE, DELETE
- ExecuteScalar - returneaza o singura valoare dintr-o interogare
- ExecuteReader - returneaza o multime rezultat, sub forma unui obiect DataReader

DataReader

- contine un recordset bazat pe conexiune, forward-only, read-only
- obiectele DataReader nu pot fi instantiat direct, sunt returnate ca rezultat al metodei ExecuteReader a unui obiect Command (SqlCommand - SqlDataReader etc)
- o singura linie din recordset este in memorie la un moment dat, deci se foloseste un minim de resurse, dar este necesara mentinerea activa a unui obiect Connection pe durata de viata a obiectului DataReader

DataAdapter este clasa din nucleul tehnologiei ADO.NET, bazata pe mecanismul non-conexiune.

- faciliteaza comunicarea intre baza de date si DataSet
- populeaza obiectele DataTable sau DataSet ori de cate ori se apeleaza metoda Fill
- metoda Update inregistreaza modificarile, efectuate local, in baza de date

La apelul metodei Update, se copieaza modificarile din DataSet in baza de date, executandu-se una din comenziile reprezentate de InsertCommand, DeleteCommand sau UpdateCommand.

Exista controale folosite la afisare de date: **Labels**, **TextBoxes** – informatii unicte; dar in mod curent se folosesc si unele pentru afisarea unor colectii de informatii, cum ar fi: **ListBoxes**, **DropDownLists** si **GridViews**. Acestea din urma se folosesc foarte mult in aplicatii de management al informatiilor, cum ar fi de exemplu una de biblioteca, unde se poate dori afisarea tuturor cartilor dintr-un anumit domeniu, etc.

4.4.1. Obiectele ADO.Net

4.4.1.1 *SqIConnection*

Un obiect *SqlConnection* este la fel ca orice alt obiect C#. de cele mai multe ori declararea si instantierea se face in acelasi timp:

```
SqlConnection sqlConn = new SqlConnection("Data Source=(local);Initial Catalog=Northwind;Integrated Security=SSPI");
SqlConnection sqlConn1 = new SqlConnection("Data Source=DatabaseServer;Initial Catalog=Northwind;User ID=YourUserID;Password=YourPassword");
OleDbConnection oleDbConn = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=MyDatabase.mdb");
```

Obiectul *SqlConnection* de mai sus este instantiat folosind un constructor care primeste ca parametru un string. Acest argument este stringul de conectare.

Scopul instantierii unui obiect de tip *SqlConnection* este ca alte obiecte ADO.Net sa poata lucra cu baza de date. Alte obiecte, cum ar fi *SqlDataAdapter* si *SqlCommand*, au constructori care primesc obiectul conexiune ca parametru. Atunci cand se lucreaza cu o baza de date trebuie urmati pasii:

1. Instantierea unui obiect *SqlConnection*;
2. Deschiderea conexiunii;
3. Trimiterea conexiunii ca parametru altor obiecte ADO.Net;
4. Realizarea operatiunilor asupra bazei de date;
5. Inchiderea conexiunii.

4.4.1.2 *SqICommand*

Obiectele de tipul *SqlCommand* permit specificare tipului de actiune asupra bazei de date. De exemplu se poate face o interogare, inserare, modificare sau stergere.

Atunci cand se realizeaza o interogare in baza de date, se obtine un tabel rezultat care trebuie sa poata fi vizualizat. Pentru a obtine acest lucru folosind obiecte *SqlCommand* este folosita metoda *ExecuteReader* care intoarce un obiect de tipul *SqlDataReader*. Exemplul de mai jos arata modul de obtinere a unei instante *SqlDataReader*.

```
SqlCommand cmd = new SqlCommand ("SELECT CategoryName FROM Categories",
conn);
```

```
SqlDataReader rdr = cmd.ExecuteReader();
```

Pentru a insera valori intr-o baza de date trebuie apelata functia *ExecuteNonQuery* pe un obiect *SqlCommand*. Exemplul urmator arata modul de inserare intr-o baza de date.

```
string insertString = @"INSERT INTO Categories (CategoryName, Description)
VALUES ('Miscellaneous', 'Whatever doesn't fit elsewhere');
SqlCommand cmd = new SqlCommand (insertString, conn);
cmd.ExecuteNonQuery();
```

Modificarea si stergerea datelor dintr-o baza de date se face la fel ca si inserarea, dar ca punem primul parametru al constructorului *SqlCommand* pe valoarea corespunzatoare.

Uneori avem nevoie dintr-o baza de date doar de o singura valoare, care poate fi suma, media, etc. inregistrarilor dintr-un tabel. Apeland *ExecuteReader* si apoi calculand acea valoare in program nu este cea mai eficienta metoda de a ajunge la rezultat. Cea mai buna metoda este sa lasam baa de date sa faca ceea ce este necesar si sa intoarcă o singura valoare. Acest lucru il face metoda *ExecuteScalar*:

```
SqlCommand cmd = new SqlCommand ("SELECT count(*) FROM Categories", conn);
```

```
int count = (int) cmd.ExecuteScalar();
```

4.4.1.1.3 SqlDataReader

Tipul SqlDataReader este folosit pentru a citi date in cea mai eficienta metoda posibila. NU poate fi folosit pentru scriere. O data citita o informatie nu mai poate fi citita inca o data. SqlDataReader citeste sequential date.

Datprita faptului ca citeste doar inainte (forward-only) permite acestui tip de date sa fie foarte rapid in citire. Overhead-ul asociat este foarte mic (overhead generat cu inspectarea rezultatului si a scrierii in baza de date). Daca intr-o aplicatie este nevoie doar de informatii care vor fi citite o singura data, sau rezultatul unei interogari este prea mare ca sa fie retinut in memorie (caching) SqlDataReader este solutia cea mai buna.

Obtinerea unei instante de tipul SqlDataReader este putin diferita de instantierea normala - trebuie apelata metoda ExecuteDataReader. Daca pentru instantiere este folosit operatorul new veti obtine un obiect cu care nu puteti face nimic pentru ca nu are o conexiune si o comanda atasate.

SqlDataReader obtine datele intr-un stream sequential. Pentru a citi aceste informatii trebuie apelata metoda Read; aceasta citeste un singur rand din tabelul rezultat. Metoda clasica de a citi informatia dintr-un SqlDataReader este de a itera intr-o bucla while asa cum se vede in figura 4 la liniile 32-35.

Metoda Read intoarce true cat timp mai este ceva de citit din stream.

4.4.1.1.4

4.4.1.1.5 Instalarea unei aplicatii web pe serverul IIS

Evident ca dupa dezvoltarea unei aplicatii, ea trebuie livrata beneficiarului. Pentru ca ea sa meargă, este nevoie sa fie "depusa" pe serverul acestuia de web. In primul rand, fisierele trebuie copiate undeva pe hard-disk-ul serverului web (nu neaparat in calea predefinita ...\\Inetpub\\wwwroot\). Dupa aceea, in IIS-ul server-ului web, trebuie creat un nou director virtual, care sa indice catre calea directorului fizic unde se afla de fapt aplicatia.

Deci, dupa ce ati depus proiectul ASP.NET pe serverul web (asta se face prin simpla copiere), mergeti in IIS, click dreapta pe Default Web Site, New -> Virtual Directory, alegeti un Alias (acesta va fi de fapt numele aplicatiei asa cum va fi ea vazuta din afara), navigati la calea directorului propriu-zis si ... gata. Acum, aplicatia va putea fi referita din exterior cu numele: http://<nume_server>/<alias> ; unde <nume_server> este de fapt situl principal expus de serverul web, iar <alias> este ce ati ales voi la crearea directorului virtual in IIS.

Evident, trebuie neaparat sa aveți grija de securizarea aplicatiei daca este cazul, dupa cum se vede si in paragraful precedent.

4.4.2. Configurare, mentenanta

Avantajele folosirii unui fisier web.config. In primul si in primul rand, sa ne imaginam o aplicatie de E-Commerce care lucreaza cu un server de baze de date si sa ne imaginam ca noi facem aplicatia si o dam la mai multi clienti.

In mod evident, serverul de baze de date se va numi in mod diferit la fiecare client. Cum facem deci sa putem livra aplicatia clientilor, fara a recompila codul la fiecare ? Cum facem conexiunea la baza de date pentru fiecare client ? Ei bine, folosim web.config :

```
<appSettings>
    <add key="Database" value="Data Source=MISHU\MISHUSQL;Initial
Catalog=elearning;User Id=***;Password=***" />
</appSettings>
```

Dupa cum se poate vedea aici, am pus o inregistrare care contine chiar un string de conexiune la baza de date. Cum fac conexiunea in aplicatia mea ? Caut in web.config o cheie cu numele « Database ». Iata:

```
public static SqlConnection ConnectToDB()
{
    string strParameters;
    System.Configuration.AppSettingsReader      apsr      =      new
System.Configuration.AppSettingsReader();
    strParameters = (string)apsr.GetValue("Database",typeof(string));
    SqlConnection sqlConn = new SqlConnection(strParameters);
    return sqlConn;
}
```

5. Referinte

http://www.sei.cmu.edu/str/descriptions/clientserver_body.html
http://en.wikipedia.org/wiki/Multitier_architecture
http://en.wikipedia.org/wiki/Service-oriented_architecture
<http://en.wikipedia.org/wiki/Model-view-controller>
http://en.wikipedia.org/wiki/OSI_Model
http://en.wikipedia.org/wiki/Internet_protocol_suite
http://www.w3schools.com/web/web_scripting.asp
<http://en.wikipedia.org/wiki/HTTP>
<http://www.w3.org/TR/html401/>
<http://www.w3schools.com/html/default.asp>
<http://www.w3.org/Style/CSS/>
<http://www.w3schools.com/css/default.asp>
http://en.wikipedia.org/wiki/.NET_Framework
<http://www.charlespetzold.com/dotnet/>

Proiectarea, organizarea și evaluarea activităților didactice

Cuprins

<i>I. Programa școlară reper obligatoriu pentru un demers flexibil.....</i>	<i>93</i>
<i>II. Aplicarea programelor școlare.....</i>	<i>94</i>
<i>III. Proiectarea demersului didactic.....</i>	<i>95</i>
III.1. Lectura personalizată a programelor școlare	
III.2. Planificarea calendaristică	
III.3. Proiectarea unei unități de învățare	
III.4. Proiectarea activității de evaluare	
III.4.1. Tipuri de itemi	
III.4.2. Metode complementare de evaluare	
<i>IV. Proiectul unității de învățare - Proiectul de lecție ?.....</i>	<i>107</i>
<i>V. Anexe</i>	<i>108</i>

1. Planificări calendaristice și Proiecte de unități de învățare

I. Programa școlară reper obligatoriu pentru un demers flexibil

Programa școlară este parte a Curriculumului național. Termenul de curriculum derivă din limba latină unde, printre altele înseamna *drum către*.

Programa școlară descrie oferta educațională a unei anumite discipline pentru un parcurs școlar determinat Filosofia contemporană a educației a evidențiat diferența dintre o educație bazată pe **curriculum**, adică având ca element central la toate etajele sale *activitatea de proiectare și programa analitică*, care are în centrul activității didactice ideea de *programare* a traseului elevului către un țel cunoscut și impus doar de către adulți.

Conceptual, **programele școlare actuale**, se diferențiază de „programele analitice” prin accentul pe care îl pun pe interiorizarea unui mod de gândire specific fiecărui domeniu transpus în școală prin intermediul unui obiect de studiu, nu pe succesiunea conținuturilor și pe numărul de ore alocate lor.

Actualele programe școlare subliniază importanța rolului reglator al achizițiilor elevilor în plan **formativ**. Centrarea pe competențe reprezintă modalitatea care face ca sintagma **centrarea pe elev** să nu rămână o lozincă fără conținut.

Proiectarea curriculumului pe competențe vine în întâmpinarea cercetărilor din psihologia cognitivă, conform cărora prin competență se realizează în mod exemplar transferul și mobilizarea cunoștințelor și a deprinderilor în situații/contexte noi și dinamice.

Modelul de proiectare curriculară centrat pe competențe simplifică structura curriculumului și asigură o mai mare eficiență a proceselor de predare/învățare și evaluare. Acesta permite operarea la toate nivelurile cu aceeași unitate: competența, în măsură să orienteze demersurile agenților implicați în procesul de educație:

- conceptori de curriculum;
- specialiștii în evaluare;
- profesorii;
- inspectorii;
- elevii;
- părinții.

Fără a intra în detalii conceptuale, formulăm câteva definiții de lucru necesare pentru explicarea manierei în care au fost concepute programele școlare.

Definim **competențele** ca fiind ansambluri structurate de cunoștințe și deprinderi dobândite prin învățare; acestea permit identificarea și rezolvarea în contexte diverse a unor probleme caracteristice unui anumit domeniu.

Structura programei școlare cuprinde: o notă de prezentare, competențe generale, competențe specifice și conținuturi, valori și atitudini, sugestii metodologice.

- ✓ **Competențele generale** se definesc pe obiect de studiu și se formează pe durata învățământului liceal. Ele au un grad ridicat de generalitate și complexitate și au rolul de a orienta demersul didactic către achizițiile finale ale elevului.
- ✓ **Competențele specifice** se definesc pe obiect de studiu și se formează pe parcursul unui an școlar. Ele sunt derivate din competențele generale, fiind etape în dobândirea acestora. Competențelor specifice li se asociază prin programă unități de conținut.

Componenta fundamentală a programei este cea referitoare la competențe specifice și conținuturi.

Pentru a asigura o marjă cât mai largă de acoperire a obiectelor de studiu, s-a pornit de la o diferențiere cât mai fină a **etapelor unui proces de învățare**. Acestea le corespund categorii de competențe organizate în jurul câtorva **verbe definitorii**, ce exprimă complexe de operații mentale:

1. **Receptarea** - concretizată prin următoarele concepte operaționale:
 - identificarea de termeni, relații, procese;
 - observarea unor fenomene, procese;
 - perceperea unor relații, conexiuni;
 - nominalizarea unor concepte;
 - culegerea de date din surse variante;
 - definirea unor concepte.
2. **Prelucrarea primară** (a datelor) - concretizată prin următoarele concepte operaționale:
 - compararea unor date, stabilirea unor relații;
 - calcularea unor rezultate parțiale;
 - clasificarea datelor;
 - sortarea-discriminarea;
 - investigarea, descoperirea, explorarea;
 - reprezentarea unor date;
 - experimentarea.
3. **Algoritmizarea** - concretizată prin următoarele concepte operaționale:
 - reducerea la o schemă sau model;
 - anticiparea unor rezultate;

- remarcarea unor invariante;
- reprezentarea datelor

- rezolvarea de probleme prin modelare si algoritmizare.

4.Exprimarea - concretizata prin urmatoarele concepte operaționale:

- descrierea unor stări, sisteme, procese, fenomene;
- generarea de idei;

- argumentarea unor enunțuri;
- demonstrarea.

5.Prelucrarea secundară (a rezultatelor) – concretizată prin urmatoarele concepte operaționale:

- compararea unor rezultate, date de ieșire, concluzii;
- calcularea, evaluarea unor rezultate;
- interpretarea rezultatelor;
- analiza de situații;

- elaborarea de strategii;
- relaționări între diferite tipuri de reprezentări, între reprezentare și obiect.

6.Transferul, care poate fi concretizat prin urmatoarele concepte operaționale:

- aplicarea în alte domenii;
- generalizarea și particularizarea;
- integrarea unor domenii;
- verificarea unor rezultate;
- optimizarea unor rezultate;

- transpunerea într-o altă sferă;
- negocierea;
- realizarea de conexiuni între rezultate;
- adaptarea și adevararea la context.

Competențele generale ce se urmăresc să fie formate la elevi pe parcursul treptei liceale de școlarizare precum și competențele specifice fiecărui an de studiu, derivate din acestea, se stabilesc pornind de la modelul de generare prin gruparea categoriilor de concepte operaționale în funcție de dominantele avute în vedere.

- ✓ **Valorile și atitudinile** apar în mod explicit sub forma unei liste separate în programa fiecărui obiect de studiu. Ele acoperă întreg parcursul învățământului liceal și orientează dimensiunile *axioiologică și afectiv-atitudinală* aferente formării personalității din *perspective fiecărei discipline*. Realizarea lor concretă derivă din activitatea didactică permanentă a profesorului, constituind un implicit al acesteia. Valorile și atitudinile au o importanță egală în reglarea procesului educativ ca și competențele - care acoperă dimensiunea cognitiva a personalității - dar se supun altor criterii de organizare didactico-metodică și de evaluare.
- ✓ **Sugestiile metodologice** cuprind recomandări generale privind metodologia de aplicare a programelor. Aceste recomandări se pot referi la:
 - desfasurarea efectivă a procesului de predare/învățare, centrat pe formarea de competențe;
 - identificarea celor mai adecvate metode și activități de învățare;
 - dotări/materiale necesare pentru aplicarea în condiții optime a programelor;
 - evaluarea continuă.

Dincolo de structura unitară a programelor școlare, curriculumul național actual propune o **ofertă flexibilă**, ce permite profesorului **adaptarea** cadrului formal la **personalitatea sa și la specificul clasei de elevi** cu care lucrează. Elementele care asigură acest **reglaj** sunt:

- posibilitatea intervenției profesorului în succesiunea elementelor de conținut, cu condiția asigurării coerentei tematice și a respectării logicii interne a domeniului;
- lipsa prescrierii de la centru a intervalului de timp alocat elementelor de conținut;
- posibilitatea modificării, a completării sau a înlocuirii activităților de învățare, astfel încât acestea să permită un demers didactic personalizat.

II.Aplicarea programelor școlare

Existența unor programe centrate pe achizițiile elevilor determină un anumit sens al schimbării în didactica fiecărei discipline. Tabelul de mai jos prezintă în antiteză caracteristici ale procesului de predare-învățare din didactica tradițională și didactica actuală.

Criterii	Strategii didactice	
	Orientare tradițională	Orientare modernă
Rolul elevului	Urmărește prelegerea, expunerea, explicația profesorului.	Exprimă puncte de vedere proprii.
	Încearcă să rețină și să reproducă ideile auzite	Realizează un schimb de idei cu ceilalți.
	Acceptă în mod pasiv ideile transmise.	Argumentează; pune și își pune întrebări cu scopul de a înțelege, de a realiza sensul unor idei

	Lucrează izolat.	Cooperează în rezolvarea problemelor și a sarcinilor de lucru.
Rolul profesorului	Expune, ține prelegeri.	Facilitează și moderează învățarea.
	Impune puncte de vedere.	Ajută elevii să înțeleagă și să explice punctele de vedere proprii.
	Se consideră și se manifestă în permanență „ca un părinte”.	Este partener în învățare.
Modul de realizare a învățării	Învățarea are loc predominant prin memorare și reproducere de cunoștințe, prin apel doar la exemple „clasice”, validate.	Învățarea are loc predominant prin formare de competențe și deprinderi practice.
	Învățarea conduce la competiție între elevi, cu scopul de ierarhizare	Învățarea se realizează prin cooperare.
Evaluarea	Vizează măsurarea și aprecierea cunoștințelor (ce știe elevul).	Vizează măsurarea și aprecierea competențelor (ce poate să facă elevul cu ceea ce știe).
	Pune accent pe aspectul cantitativ (cât de multă informație deține elevul).	Pune accent pe elementele de ordin calitativ (valori, atitudini).
	Vizează clasificarea „statică” a elevilor,	Vizează progresul în învățare pentru fiecare elev.

De fapt, diferența dintre didactica tradițională și cea actuală constă în **modul de concepere și organizare a situațiilor de învățare** (riguroș dirijate în primul caz și având autonomie de diferite grade, în cel de-al doilea). Altfel spus, o strategie este legitimă sau ilegitimă nu în general, ci potrivit unor circumstanțe concrete; **profesorul eficient** este acela care știe:

- o să selecționeze
- o să combine
- o să varieze diferite metode
- o să aleagă strategii adecvate.

III. Proiectarea demersului didactic

Predarea reprezintă activitatea profesorului de organizare și conducere a ofertelor de învățare care are drept scop facilitarea și stimularea învățării eficiente la elevi.

Proiectarea demersului didactic este acea activitate desfășurată de profesor care constă în anticiparea etapelor și a acțiunilor concrete de realizare a predării. Proiectarea demersului didactic presupune:

- lectura personalizată a programelor școlare;
- planificarea calendaristică;
- proiectarea secvențială (a unităților de învățare).

III.1. Lectura personalizată a programelor școlare

În contextul noului curriculum, conceptul central al proiectării didactice este demersul didactic personalizat, iar instrumentul acestuia este **unitatea de învățare**. Demersul personalizat exprimă dreptul profesorului - ca și al autorului de manual - de a lua decizii asupra modalităților pe care le **consideră** optime în creșterea **calității** procesului de învățamant, respectiv, **răspunderea personală** pentru a **asigura** elevilor un parcurs școlar **individualizat**, în funcție de condiții și cerințe concrete.

LECTURAREA PROGRAMEI se realizează “pe orizontală” în succesiunea următoare:



III.2. Planificarea calendaristică

În contextul noului curriculum, **planificarea calendaristică** este un document administrativ care asociază elemente ale programei cu alocarea de timp considerată optimă de către profesor pe parcursul unui an școlar.

În elaborarea planificărilor calendaristice recomandăm parcurgerea următoarelor etape:

1. Realizarea asocierilor dintre competențele specifice și conținuturi.
2. Împărțirea în unități de învățare.
3. Stabilirea succesiunii de parcurgere a unităților de învățare.

4. Alocarea timpului considerat necesar pentru fiecare unitate de învățare, în concordanță cu competențele specifice și conținuturile vizate.

Planificările pot fi întocmite pornind de la următoarea rubrică

Unitatea școlară.....

Disciplina.....

Profesor.....

Clasa/Nr. ore pe săpt./

Planificare calendaristică

Anul școlar.....

Unități de învățare	Competențe specifice	Conținuturi	Nr. de ore alocate	Săptămâna	Observații

In acest tabel:

- Unitățile de învățare se indică prin titluri (teme) stabilite de către profesor.
- In rubrica Competențe specifice se trec simbolurile competențelor specifice din programa școlară.
- Continuturile selectate sunt cele extrase din lista de continuturi ale programei.
- Numarul de ore alocate se stabilește de către profesor în funcție de experiența acestuia și de nivelul de achiziții ale elevilor clasei.

Întregul cuprins al planificării *are valoare orientativă*, eventualele modificări determinate de aplicarea efectivă la clasă putând fi consimilate în rubrica Observații.

O planificare anuală corect întocmită trebuie să acopere integral programa școlară la nivel de competențe specifice și conținuturi.

III.3. Proiectarea unei unități de învățare

Elementul generator al planificării calendaristice este unitatea de învățare.

O unitate de învățare reprezintă o structură didactică deschisă și flexibilă, care are următoarele caracteristici:

- determină formarea la elevi a unui comportament specific, generat prin integrarea unor competențe specifice;
- este unitară din punct de vedere tematic;
- se desfășoară în mod sistematic și continuu pe o perioadă de timp;
- se finalizează prin evaluare.

Detaliem în continuare elementele esențiale ale proiectării unei unități de învățare.

Se recomandă utilizarea următorului format.

Unitatea școlară.....

Disciplina.....

Profesor.....

Clasa/Nr. ore pe săpt./

Proiectul unității de învățare.....

Nr. de ore alocate.....

Conținuturi	Competențe specifice	Activități de învățare	Resurse	Evaluare	Observații

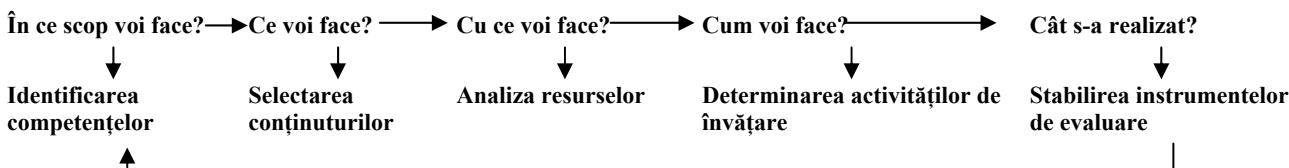
- Conținuturi: apar inclusiv detalieri de conținut necesare în explicitarea anumitor parcursuri, respectiv în cuplarea lor la baza proprie de cunoastere a elevilor.
- In rubrica Competențe specifice se trec simbolurile competențelor specifice din programa școlară
- Activitățile de învățare pot fi cele din programa școlară, completate, modificate sau chiar înlocuite de altele, pe care profesorul le consideră adecvate pentru atingerea obiectivelor propuse.
- Rubrica Resurse cuprinde specificări de timp, de loc, forme de organizare a clasei, material didactic folosit etc.
- În rubrica Evaluare se menționează instrumentele sau modalitățile de evaluare aplicate la clasă.

Finalul fiecărei unități de învățare presupune **Evaluare sumativă**.

Desi denumirea si alocarea de timp pentru unitățile de învățare se stabilesc la începutul anului școlar prin planificare, este recomandabil ca proiectele unităților de învățare să se completeze ritmic pe parcursul anului, având în avans un interval de timp optim pentru ca acestea să reflecte cât mai bine realitatea.

In completarea rubricăției, se urmarește corelarea elementelor celor cinci coloane. Practic pe baza indicațiilor din planificare se fac detalierile pe orizontală, ordonând activitățile în succesiunea derulării, raportandu-le la competențe și specificând resursele necesare bunei desfășurări a procesului didactic.

Proiectarea unității de învățare - ca și a lecției - începe prin parcurgerea schemei urmatoare:



Conceptul de unitate de învățare are rolul să materializeze conceptul de demers didactic personalizat, flexibilizând proiectarea didactică și definind în acest sens pentru practica didactică premise mai bine fundamentate din punct de vedere pedagogic.

Identificarea unei unități de învățare se face prin tema acesteia. Stabilirea temei de către profesor pe baza lecturii programei, utilizând surse diverse, este primul pas în identificarea unităților de învățare în care va fi imparțită materia anului școlar, respectiv, în organizarea unu demers didactic personalizat. Temele sunt enunțuri complexe legate de analiza scopurilor învățării, formulări fie originale, fie preluate din lista de conținuturi a programei, sau din manual formulări care reflectă din partea profesorului o înțelegere profundă a scopurilor activității sale, talent pedagogic, inspirație, creativitate.

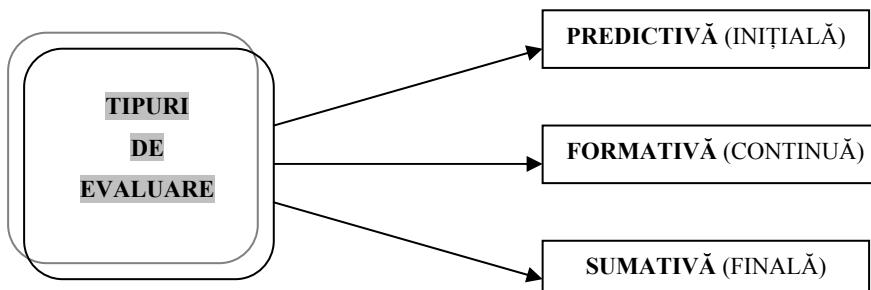
Activitățile de învățare se construiesc prin corelarea competențelor cu conținuturile și presupun orientarea către un anumit scop, redat prin tema activității. În momentul propunerii lor spre rezolvare elevilor, activitățile de învățare vor fi transpușe într-o anumită formă de comunicare inteligeabilă nivelului de vârstă.

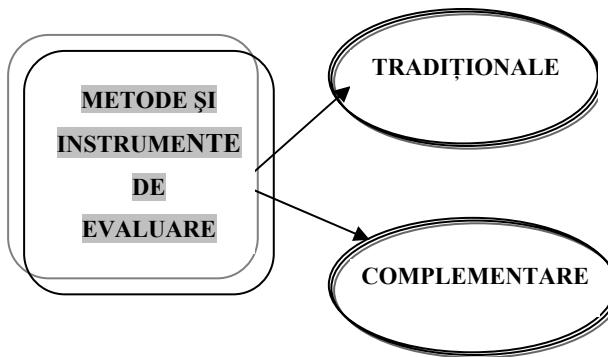
Intr-o abordare pragmatică, **resursele** cuprind acele elemente care asigură cadrul necesar pentru buna desfășurare a activităților de învățare. Astfel, profesorul va menționa în această rubrică forme de organizare a clasei (tipuri de interacțiuni ale resurselor umane), mijloace de învățământ, alocarea de timp, precum și orice alte elemente pe care le consideră utile în derularea scenariului didactic.

In condițiile noului curriculum, lectura programei și a manualelor nu mai este în mod obligatoriu liniară. Programa trebuie parcursă în mod necesar de către toți, dar ea, ca și manualele se pliază unei citiri personale și adaptate. Asupra conținuturilor programei profesorul poate interveni prin regruparea lor sub teme de unități de învățare pe care le-a stabilit.

III.4. Proiectarea activității de evaluare se realizează concomitent cu proiectarea demersului de predare/învățare și în deplină concordanță cu acesta. Câteva întrebări utile în proiectarea instrumentelor de evaluare sunt urmatoarele:

- Care sunt competențele din programa școlară, pe care trebuie să le dobândească elevii?
- Care sunt performanțele minime, medii și superioare pe care le pot atinge elevii, pentru a demonstra că au atins aceste competențe?
- Care este specificul colectivului de elevi pentru care îmi propun evaluarea?
- Când și în ce scop evaluez?
- Pentru ce tipuri de evaluare optez?
- Cu ce instrumente voi realiza evaluarea?
- Cum voi proceda pentru ca fiecare elev să fie evaluat prin tipuri de probe cât mai variate astfel încât evaluarea să fie cât mai obiectivă?
- Cum voi folosi datele oferite de instrumentele de evaluare administrate, pentru a elimina eventualele blocaje constatate în formarea elevilor și pentru a asigura progresul scolar al fiecaruia dintre ei?





- Probe scrise
- Probe orale
- Probe practice
- Observarea sistematică a elevilor
- Investigația
- Proiectul
- Portofoliul
- Tema pentru acasă
- Tema de lucru în clasă
- Autoevaluarea

Adeseori, evaluarea formativă este înlocuită cu evaluarea curentă, în accepțiunea tradițională de “notare ritmică”, constând în probe stereotipe precum dictări, calcule, compuneri, diverse tipuri de exerciții etc., care sunt apreciate prin note în mod aproximativ, fără interes în ceea ce privește nivelul cognitiv, afectiv, psihomotor, relațional. O astfel de evaluare are în vedere doar unele tipuri de comportamente, de obicei nerelevante pentru personalitatea elevului și neglijeează aspecte importante ca: gândirea, imaginația, atitudinea de responsabilitate, adoptarea unor metode proprii de lucru, competența de comunicare și de relaționare etc.

Ceea ce este generalizat în evaluarea curentă din școli este stocarea în memorie a abstracțiilor și promptitudinea reproducерii acestora. Multe cunoștințe și capacitați care necesită evaluare sunt ignorate, deși sunt, în egală măsură, variabile importante ale învățării, cum ar fi: atitudinea elevului față de învățare, față de disciplina de studiu, față de educator și față de interrelațiile din interiorul colectivului de elevi, modul în care elevul învăță (sistematic sau sporadic), modul cum abordează cunoștințele pentru a rezolva probleme practice, specifice vieții cotidiene etc.

Focalizată pe unitatea de învățare, evaluarea ar trebui să asigure evidențierea progresului înregistrat de elev în raport cu sine însuși, pe drumul atingerii competențelor prevăzute în programă. Este important să fie evaluată nu numai cantitatea de informație de care dispune elevul, ci, mai ales, ceea ce poate el să facă utilizând ceea ce știe sau ceea ce intuiște.

In acest sens, câteva aspecte pot fi avute în vedere:

- modificarea raportului dintre *evaluarea sumativă*, care inventariază, selectează și ierarhizează prin notă și *evaluarea formativă*, ce are drept scop valorificarea potențialului de care dispun elevii și conduce la perfecționarea continuă a stilului și a metodelor proprii de învățare;
- realizarea unui echilibru dinamic între *evaluarea scrisă* și *evaluarea orală*: aceasta din urmă, deși presupune un volum mare de timp pentru aprecierea tuturor elevilor și bloage datorate emoției sau timidității, prezintă avantaje deosebite, ca: realizarea interacțiunii elev-profesor, demonstrarea comportamentului comunicativ și de inter-relaționare a elevului.
- folosirea cu mai mare frecvență a metodelor de autoevaluare și evaluare prin consultare în grupuri mici.

În raport cu momentele realizării evaluării, în **proiectul unității de învățare** apar specificații: **evaluare inițială, formativă sau sumativă**.

Fiecare activitate de evaluare a rezultatelor școlare trebuie însoțită în mod sistematic de o autoevaluare a procesului pe care profesorul l-a desfășurat cu toți elevii și cu fiecare elev. Numai astfel pot fi stabilite modalitățile prin care poate fi reglată, de la o etapă la alta, activitatea de învățare / formare a elevilor în mod diferențiat.

III.4.1. Tipuri de itemi

Item = <întrebare> + <formatul acesteia> + <răspunsul așteptat>

Teoria și practica evaluării evidențiază mai multe criterii pe baza cărora pot fi clasificați itemii. Unul dintre criteriile cel mai des utilizate este acela al **gradului de obiectivitate oferit în corectare**. În funcție de acest criteriu, itemii pot fi clasificați în trei mari categorii:

a. **Itemii obiectivi** asigură un grad de obiectivitate ridicat în măsurarea rezultatelor școlare și testează un număr mare de elemente de conținut într-un interval de timp relativ scurt. Răspunsul așteptat este bine determinat, ca și modalitatea de notare a acestuia.

b. **Itemii semiobiectivi** permit ca răspunsul așteptat să nu fie totdeauna unic determinat, modalitatea de corectare și notare inducând uneori mici diferențe de la un corector la altul. Aceștia testează o gamă mai variată de capacitați intelectuale, oferind în același timp posibilitatea de a utiliza și materiale auxiliare în rezolvarea sarcinilor de lucru propuse.

c. **Itemii subiectivi (cu răspuns deschis)** solicită un răspuns amplu, permitând valorificarea capacitaților creative ale elevilor. Aceștia sunt relativ ușor de construit, principala problemă constituind-o modul de elaborare a schemei de notare astfel încât să se poată obține unitate și uniformitate la nivelul corectării.

a. Itemi obiectivi

Fie că este vorba de activități de proiectare și programare (Informatică), fie că este vorba de activități de utilizare și combinare a instrumentelor informaticice (Tehnologia Informației), lucrul cu calculatorul implică formulări standardizate, lipsite de echivoc, itemii obiectivi și cei semiobiectivi reprezentând instrumente de evaluare fravent aplicate la aceste discipline.

Itemii obiectivi sunt caracterizați prin:

- structurarea sarcinilor propuse și standardizarea formatului lor de prezentare;
- corelarea strictă a sarcinilor cu obiectivele de evaluării;
- capacitatea de a testa un număr mare de elemente de conținut într-un timp relativ scurt;
- obiectivitate privind aprecierea răspunsului;
- posibilitatea asocierii cu un sistem de notare extrem de simplu: punctajul aferent se acordă integral, se acordă parțial conform unei reguli (formule) de calcul sau nu se acordă deloc (în funcție de răspunsul așteptat);
- rolul secundar pe care îl au de a obișnui elevul cu formulări standard, științifice, elemente utile în construirea răspunsurilor pentru itemii semiobiectivi și subiectivi.

a.1. Itemi cu alegere duală

Alegerea duală presupune formularea unei cerințe cu două variante complementare de răspuns (*Adevărat/Fals, Da/Nu, Corect/Incorect* etc.).

Se pot verifica prin intermediu itemilor cu alegere duală:

- cunoștințele legate de corectitudinea sintactică a unor expresii (comenzi, instrucțiuni, notații etc.);
- înțelegerea semnificației unor noțiuni din terminologia de specialitate (denumiri, instrumente de prelucrare, metode de rezolvare, proprietăți etc.)
- recunoașterea unor explicații, definiții, sau imagini.

Itemii de acest tip se prezintă sub forma unor întrebări sau enunțuri, efortul elevului reducându-se la identificarea unui răspuns din două posibile. Achizițiile evaluate prin itemii cu alegere duală sunt de regulă rudimentare. Fac excepție enunțurile care pun în evidență justificări ale unor proprietăți, operații sau reguli, justificări care necesită achiziții cognitive superioare. Tot în categoria itemilor cu alegere duală se pot realiza cerințe care necesită din partea elevului operații de anticipare a efectului unei operații prin aplicarea unui sistem riguros de cunoștințe într-un context nou. Aceștia sunt itemii cu cel mai înalt grad de dificultate. Atragem atenția asupra faptului că lipsa “firului roșu” care pune în evidență elementele sistematice întâlnite în utilizarea unui produs soft (chiar și în cazul unui soft de bază) conduc la un experimentalism accentuat care împiedică formarea capacitații elevului de a se adapta situațiilor noi, neexersate.

Factorul de discriminare fiind însă extrem de mic, elevul va putea obține un rezultat acceptabil la un test format numai din astfel de itemi alegând la întâmplare un răspuns dintre cele două admise pentru fiecare item în parte. De obicei, itemii cu alegere duală sunt formulați în combinație cu itemi subiectivi de tipul “Justificați...”, “Scriți varianta corectă...”, “Explicați în ce constă eroarea...” etc. În aceste cazuri, o parte din punctaj este alocată justificării.

Pentru proiectarea corectă a itemilor cu alegere duală este necesară respectarea următoarelor cerințe:

- formularea clară a enunțului, fără ambiguități sau formulări incomplete;
- dacă se solicită aprecierea cu ADEVĂRAT/FALS, se vor evita enunțurile foarte generale;
- selectarea unor enunțuri relevante pentru domeniul de cunoaștere sau categoria de competențe testată (uneori, efortul de a realiza enunțuri fără echivoc duce la elaborarea de itemi nesemnificativi din punct de vedere educațional sau științific);
- se va evita utilizarea unor enunțuri negative, acestea conduce la raționamente ce folosesc dubla negație, inducând un grad înalt de ambiguitate;

- se vor evita enunțurile lungi și complexe, prin eliminarea elementelor redundante inutile în raport cu ideea enunțului și cerința itemului; nu se va folosi un limbaj academic, o terminologie foarte specializată sau o construcție lingvistică stufoasă și greoie;
- se va evita introducerea a două idei într-un singur enunț, cu excepția cazului în care se dorește evidențierea relației dintre acestea;
- enunțurile vor fi aproximativ egale ca lungime;
- enunțurile adevărate sau false să fie aproximativ egale ca număr, dar nu exact egale, deoarece acesta ar putea constitui un indiciu după care elevul încearcă să ghicească răspunsul corect.

a.2. Itemi de tip pereche

Itemii de tip pereche solicită stabilirea unor corespondențe între informațiile distribuite pe două coloane. Prima coloană conține informații de tip enunț (*premise*), cea de-a doua coloană conținând informații de tip răspuns. Elevului i se solicită să asocieze fiecare enunț cu un unic răspuns.

Cele două coloane sunt precedate de instrucțiuni de asociere în care i se explică elevului tehnica de formare a perechilor (să unească printr-o linie, să descrie perechile asociate sau doar elementele lor de identificare etc.) și se precizează dacă un răspuns poate fi folosit la mai mult de un enunț (dacă funcția de asociere este injectivă sau nu), eventual dacă există răspunsuri care nu vor fi folosite niciodată (dacă funcția de asociere este surjectivă sau nu).

Se verifică prin intermediul itemilor de tip pereche capacitatea elevului de a stabili corelații între:

- funcții și instrumente;
- simboluri și concepte;
- termeni și definiții;
- probleme și metode de rezolvare.

Itemii de acest tip permit abordarea unui volum mare de informație într-un interval de timp relativ redus. Factorul de discriminare este ceva mai mare decât în cazul itemilor cu alegere duală, strategia de asociere “la întâmplare” neconducând decât în situații foarte rare la un rezultat acceptabil privind rezultatul testului.

Pentru proiectarea corectă a itemilor de tip de pereche este necesară respectarea următoarelor cerințe:

- utilizarea unui material omogen, dintr-o sferă relativ restrânsă;
- utilizarea unui număr egal de premise și răspunsuri, astfel încât, dacă elevul asociază corect *n*-l enunțuri dintre cele *n* date, să nu rezulte automat răspunsul pentru cel de-al *n*-lea enunț;
- aranjarea listei de răspunsuri (mai ales dacă sunt multe) într-o ordine logică, astfel încât căutarea răspunsului în listă să se realizeze cât mai comod;
- aranjarea enunțurilor în listă astfel încât să nu se poată intui o regulă de asociere (referințele să fie “încrucișate”);
- aranjarea coloanelor astfel încât acestea să încapă în întregime pe aceeași pagină.

a.3. Itemi cu alegere multiplă

Itemii cu alegere multiplă sunt cele mai utilizate tipuri de itemi, în special în testele standardizate (bacalaureat, admitere etc.)

Un item cu alegere multiplă este format dintr-un enunț numit **premisă** sau **bază** și un număr de opțiuni din care elevul trebuie să aleagă un singur răspuns numit **cheie**. Celealte răspunsuri, neconforme cu cerința, dar plauzibile poartă numele de **distractori**.

Se verifică prin intermediul itemilor de tip pereche capacitatea elevului de a identifica:

- definiții și notații;
- instrumentul adecvat unei prelucrări;
- secvențe de program care realizează o anumită prelucrare;
- expresii cu o valoare dată;
- termeni și expresii de specialitate;
- metode de rezolvare și tehnici de implementare.

Itemii de acest tip permit abordarea unui volum mare de informație într-un interval de timp relativ redus, oferind posibilitatea evaluării unor achiziții cognitive complexe, deși nu pot măsura capacitatea elevului de a-și organiza și exprima ideile. Sunt forme de testare cu un grad mare de fidelitate, iar factorul de discriminare este mai mare decât în cazul celorlalți itemi obiectivi. Abilitatea profesorului de a elabora distractori cât mai plauzibili, care să construiască toate alternativele posibile de a greși, contribuie la reușita aplicării testelor cu alegere multiplă. Erorile comise de elevi oferă profesorului informații suplimentare necesare în autoreglarea procesului de învățământ.

O categorie de itemi cu alegere multiplă solicită răspunsul corect, celealte variante fiind greșite, în timp ce alți itemi solicită cel mai bun răspuns, pe baza unei discriminări complexe. În aceste cazuri trebuie

manifestată grija la formularea cerinței astfel încât criteriu de discriminare a "celui mai bun răspuns" să reiasă clar din enunț.

Pentru proiectarea corectă a itemilor cu alegere multiplă este necesară respectarea următoarelor cerințe:

- *stabilirea clară a cerinței, în concordanță cu obiectivul de evaluare;*
- *furnizarea tuturor informațiilor necesare în premisă, eliminându-se materialul irrelevant;*
- *formularea premisei folosind afirmații sau întrebări pozitive;*
- *construirea unor alternative plauzibile, aflate în concordanță cu premisa;*
- *construirea itemului astfel încât să existe o singură alternativă "corectă" sau "cea mai bună";*
- *construirea unor alternative astfel distractorii să fie în mod cert "greșită" sau "mai puțin buna", iar varianta cheie să fie în mod cert "corectă" sau "cea mai bună";*
- *aranjarea listei de răspunsuri într-o ordine logică, astfel încât căutarea răspunsului în listă să se realizeze cât mai comod;*
- *construirea ansamblurilor de itemi cu alegere multiplă astfel încât răspunsurile să ocupe poziții diferite în lista de variante (să nu fie în mod constant al doilea răspuns, de exemplu)*

b. Itemi semiobiectivi

Itemii semiobiectivi formează o categorie de instrumente de evaluare ce solicită construirea parțială sau totală a unui răspuns pe baza unei sarcini definite.

Itemii semiobiectivi sunt caracterizați prin:

- posibilitatea de a testa o gamă mai largă de capacitați intelectuale și rezultate ale învățării;
- crearea unor situații cognitive de nivel mai ridicat prin solicitarea de elaborare a răspunsului și nu de alegere a lui dintr-o mulțime prestabilită, ca în cazul itemilor obiectivi;
- raportarea parțial subiectivă a profesorului în raport cu răspunsul formulat (răspunsul poate fi scris ordonat sau dezordonat, formularea poate fi mai clară sau mai neclară, termenii folosiți se pot încadra în niște standarde științifice sau pot fi variante particulare ale acestora etc.)
- posibilitatea asocierii unui sistem de notare în care pot să intervină situații neprevăzute (răspunsuri neașteptate, care comportă raportări noi la barem).

b.1. Itemi cu răspuns scurt / de completare

Itemii cu răspuns scurt solicită ca elevul să formuleze un răspuns scurt sau să completeze o afirmație astfel încât aceasta să capete sens sau să aibă valoare de adevăr.

Se pot verifica prin intermediul itemilor cu răspuns scurt și de completare:

- cunoașterea unor noțiuni, expresii de specialitate, simboluri, notații etc.;
- recunoașterea și nominalizarea unor elemente vizuale specifice unui anumit mediu de lucru;
- capacitatea de integrare a unor elemente necesare din punct de vedere sintactic sau semantic într-un context dat;
- schimbarea unor elemente dintr-un context dat astfel încât să se realizeze o finalitate precizată.

Itemii cu răspuns scurt se prezintă cel mai des sub forma unor întrebări. Ei solicită un răspuns sub o formă restrânsă (un număr, un simbol, un cuvânt, o expresie, o propoziție sau frază concisă).

Itemii de completare se prezintă sub forma unui enunț, unei afirmații incomplete. Ei solicită găsirea cuvîntului sau sintagmei care completează și dă sens enunțului respectiv.

Pentru proiectarea corectă a itemilor cu răspuns scurt / de completare este necesară respectarea următoarelor cerințe:

- *formularea enunțului astfel încât să admită un răspuns scurt, exprimat cu precizie;*
- *formularea enunțului astfel încât acesta să admită un singur răspuns corect, pe cât posibil;*
- *rezervarea unor spații pentru răspuns care să sugereze numărul de cuvinte așteptate (dacă acest lucru nu reprezintă un indiciu), nu și dimensiunea lor;*
- *vizarea unui răspuns care să reprezinte o sinteză de cunoștințe sau un rezultat al înțelegерii unei situații și mai puțin o reproducere a unor informații.*

b.2. Întrebări structurate

Întrebările structurate solicită, printr-un sistem de subîntrebări relative la o temă comună, răspunsuri de tip obiectiv, răspunsuri scurte sau de completare prin care se pot evalua cunoștințele complexe referitoare la tema respectivă fără a solicita elaborarea unui răspuns deschis (eseu).

Se pot verifica prin intermediul întrebărilor structurate:

- capacitatea de a urmări, recunoaște, adapta și construi un algoritm pe o temă dată sau un program într-un limbaj de programare;

- capacitatea de a realiza din aproape în aproape o prelucrare complexă utilizând un mediu de lucru informatic. O întrebare structurată poate să conțină materiale suport și informații suplimentare ce se adaugă treptat, conferind procesului de evaluare varietate, complexitate și gradualitate. Se pot verifica totodată cunoștințe, dar și priceperi și deprinderi sporind gradul de obiectivitate în raport cu itemii cu răspuns deschis. Subîntrebările ce formează itemul permit creșterea progresivă a dificultății cerințelor, dar este recomandat ca subîntrebările să fie independente, adică răspunsul la o întrebare să nu depindă de răspunsul la întrebările precedente. Proiectarea lor necesită atenție, pricepere și timp.

Pentru proiectarea corectă a întrebărilor strucurate este necesară respectarea următoarelor cerințe:

- *redactarea subîntrebărilor astfel încât acestea să solicite răspunsuri simple la început crescând pe parcurs dificultatea acestora;*
- *formularea unor subîntrebări autoconținute (al căror răspuns corect să nu depindă de răspunsul corect la una dintre întrebările precedente); realizarea concordanței dintre enunțul general (tema întrebării) și subîntrebările formulate.*

c. Itemi subiectivi (cu răspuns deschis)

Itemii subiectivi formează o categorie de instrumente de evaluare ce vizează creativitatea elevului, originalitatea și caracterul personal al răspunsului. Deși sunt ușor de formulat, itemii subiectivi ridică probleme privind obiectivitatea evaluării.

Itemii subiectivi sunt caracterizați prin:

- abordare globală a unei sarcini asociate unui obiectiv ce nu poate fi evaluat prin intermediul itemilor obiectivi;
- crearea unor situații cognitive de nivel foarte ridicat prin solicitarea de a realiza interacțiuni reale și complexe între cunoștințe, abilități și deprinderi;
- raportarea subiectivă a profesorului în raport cu răspunsul formulat;
- necesitatea predefinirii unor criterii privind baremul de corectare și notare, criterii clare, judicioase și puternic anticipate;
- posibilitatea, în cazul în care baremul nu a prevăzut toate situațiile de interpretare și construire a răspunsului, a unor elemente noi (răspunsuri neașteptate) care comportă reanalizarea baremului.

În cazul informaticii și tehnologiei informației se pot elabora itemi subiectivi de tip eseu (structurat sau liber) și itemi de tip problemă (care necesită proiectare, redactare și uneori implementare a rezolvării).

c.1. Itemi de tip eseu

Itemii de tip eseu pot fi structurați sau liberi. Itemii structurați sunt construși astfel încât răspunsul așteptat să fie "orientat" cu ajutorul unor elemente din enunț (indicii privind ordinea de tratare, numărul de linii, formularea răspunsului, ideile care trebuie să fie atinse etc.). Un eseu liber nu furnizează în enunț nici un fel de indicații sau constrângeri, elevul având libertatea să-și strucutreze cum consideră și cum poate materialul pe care-l solicită enunțul. Acest tip de eseu comportă operații de maximă complexitate (analiză, sinteză, sistematizare și restructurare) lăsând frâu liber fantaziei și capacitaților creative ale elevului.

Deoarece la informatică elementele de creativitate se manifestă mai ales prin rezolvări de probleme și proiecte, iar în cazul tehnologiei informației prin teme practice și proiecte, itemii de tip eseu preferați sunt cei structurați, un eseu liber nefiind necesar decât rar, pentru anumite teme cu un volum mai mare de elemente "informative" în raport cu achizițiile "operatională".

Se pot verifica prin intermediul itemilor de tip eseu:

- cunoștințele globale legate de structura sistemelor de calcul, sisteme de operare, evoluția istorică a sistemelor hard și soft, principiile de utilizare a unei anumite aplicații, etapele conceptuale ale proiectării unei aplicații etc.
- capacitațile de sistematizare a unor elemente prin construirea unor scheme sau reprezentări grafice.

Itemii de tip eseu se prezintă sub forma unor cerințe generale însoțite eventual (pentru eseurile structurate) de indicii privind tratarea cerinței. Se pot adăuga restricții privind întinderea în timp sau spațiu (număr rânduri, pagini, paragrafe etc.) sau privind forma de prezentare a răspunsului.

c.2. Itemi de tip problemă

Rezolvarea unei probleme presupune soluționarea unei situații conflictuale generată de neconcordanțe ivite între sistemul de achiziții și situațiile concrete descrise de un enunț. Aceste neconcordanțe sunt generate uneori de nepotrivirea între contextul general al cunoștințelor și situațiile particulare în care acestea trebuie să fie aplicate. Alteori achizițiile sunt dobândite prin experimente (situații particulare) și, pentru a fi aplicate în cu totul alte situații, acestea trebuie să fie ridicate la un înalt nivel de abstractizare și generalizare.

Rezolvarea de probleme este o activitate specifică și des utilizată la disciplina “Informatică”, elementele gândirii algoritmice, metodele de rezolvare și tehniciile de implementare fiind supuse unui “tir” sistematic de probleme prin care acestea să formeze competențe reale de programare.

Capacitățile cognitive superioare legate de aplicare creativă, gândire divergentă, descoperirea condițiilor interne, alegerea modelului adecvat etc. sunt verificate prin itemi de acest tip. Obiectivele urmărite prin utilizarea rezolvării de probleme sunt:

- obținerea informațiilor necesare rezolvării problemei;
- formularea și testarea ipotezelor;
- descrierea metodei de rezolvare a problemei;
- elaborarea unui raport despre rezultatele obținute;
- posibilitatea de generalizare și transfer a tehniciilor de rezolvare.

Cerințe suplimentare asociate unei probleme pot pune în evidență capacitatea elevului de a estima eficiența unei rezolvări, de a construi un algoritm conform unor criterii (limita de memorie, număr de instrucțiuni etc.).

Se pot formula probleme în care se furnizează algoritmul și se cere un enunț de problemă care se rezolvă prin intermediul algoritmului respectiv. Acest tip de item impune o analiză atentă a algoritmului și asocierea lui cu una dintre problemele sau prelucrările numerice întâlnite la matematică, fizică sau în alte domenii, o formulare a enunțului care să se caracterizeze prin coerență.

Enunțurile pot fi formulate abstract, “la obiect” sau pot crea un “context” care trebuie modelat pentru a se ajunge la rezolvarea propriu-zisă. “Povestea” în spatele căreia se ascunde problema are de cele mai multe ori conotații practice, descriind situații concrete de prelucrare, amintind că rolul programatorului este acela de a “ordona” inițial informația și operațiile specifice unui anumit context și abia după aceea de a elabora algoritmul, de a implementa și verifica programul corespunzător.

Aspecte de aceeași natură se întâlnesc și în domeniul utilizării calculatorului, la Tehnologia informației, când beneficiarul formulează un enunț și utilizatorul trebuie să-și aleagă programul de aplicație adecvat, din cadrul programului să-și aleagă obiectele și instrumentele potrivite, să proiecteze, pe pași, prelucrarea astfel încât “produsul” (documentul, prezentarea, raportul etc.) să răspundă cerințelor și să fie realizat în timp record. Aspectele reale de concepție, de creativitate și gândire divergentă intervin la realizarea machetelor, a prezentărilor etc.

Evaluarea prin rezolvare de probleme la informatică ridică uneori probleme din punctul de vedere al întocmirii baremului de corectare. Unele tendințe exagerate tind să impună o corectare pe principiul: *problemă=program funcțional corect* (pornind de la premisa că “un program care aproape merge e ca un avion care aproape zboară”). Se recomandă totuși ca baremul de corectare să cuprindă fracțiuni din punctaj pentru diferitele aspecte pe care le comportă rezolvarea unei probleme la informatică: corecțitudinea sintactică, structurarea datelor și declararea variabilelor, structurarea programului, corecțitudinea algoritmului, eficiența algoritmului, tratarea unor situații limită, eventual explicarea metodei aplicate (chiar dacă a fost aplicată greșit) etc.

Se pot verifica prin intermediul itemilor de rezolvare de probleme:

- concepția unor algoritmi de rezolvare a problemelor elementare;
- asimilarea unui algoritm general prin adaptarea lui astfel încât să rezolve o problemă particulară;
- capacitatea de a alege structurile de program și de date adecvate rezolvării unei probleme;
- abilitatea de a implementa programul, de a-l depana, de a-l testa și, în funcție de erorile apărute, de a reconsidera elementele de sintaxă ale programului, strategiile de structurare a datelor sau însuși algoritmul de rezolvare (în partea practică a probei);
- capacitatea de a organiza volume mari de date cu ajutorul bazelor de date;
- discernământul în a alege un algoritm mai eficient (conform unuia dintre din criteriile studiate: număr operații, spațiu de memorie utilizat)

III.4.2. Metode complementare de evaluare

Metodele complementare de evaluare reprezintă instrumente suplimentare, nestandardizate, de evaluare dispunând de forme specifice cum ar fi: **investigația, referatul, portofoliul, proiectul, observarea sistematică a activității elevului și autoevaluarea**.

Metodele complementare realizează actul evaluării în strânsă legătură cu procesul educativ, prin întrepătrundere cu etapele acestuia, urmărind în special capacitatele cognitive superioare, motivațiile și atitudinea elevului în demersul educațional.

Metodele alternative de evaluare se caracterizează prin următoarele:

- capacitatea de a transforma relația profesor-elev înducând un climat de colaborare și parteneriat;
- posibilitatea transformării procesului de evaluare prin înlocuirea tendinței de a corecta și sănctiona prin aceea de a soluționa erorile semnalate;
- posibilitatea de a deprinde elevul cu mecanismele de autocolectare și autoeducare necesare și în procesul de integrare socială;
- utilizarea mai amplă a tehniciilor și mijloacelor didactice;
- caracterul sumativ, realizat prin evaluarea cunoștințelor, capacitaților și atitudinilor pe o periadă mai lungă de timp și dintr-o arie mai largă;
- caracterul formativ, realizat prin valorificarea atitudinii elevului în raport cu propria sa evaluare;
- capacitatea de a realiza o evaluare individualizată (observare sistematică);
- capacitatea de a educa spiritul de echipă prin activități de grup (investigații, proiecte);

- caracterul profund integrator realizat prin interdisciplinaritate, educare și instruire multilaterală.

1. Investigația

Investigația este o metodă de evaluare și învățare utilizată foarte des la disciplina Tehnologie informației și numai ocazional la disciplina Informatică.

Organizarea unei activități de evaluare și învățare prin metoda investigației presupune:

- valorificarea metodei de învățare prin descoperire;
- studiul unor documentații complementare, experimentarea unor instrumente de prelucrare nestandard, compararea și generalizarea unor tehnici și metode utilizate în tehnologie prin cercetarea altor izvoare sau prin utilizarea altor instrumente echivalente;
- extrapolarea cunoștințelor dobândite și verificarea ipotezelor formulate;
- solicitarea unor cunoștințe sau deprinderi dobândite la alte discipline prin adaptarea creațoare a acestora la cerințele temei de investigație.

În cele mai multe dintre cazuri investigația trebuie să fie organizată ca muncă independentă depusă de elev, dirijată și sprijinită de profesor. Tehnologia informației, cu multitudinea de aplicații software din domenii atât de variate, este un teren ideal pentru desfășurarea investigațiilor. Elevul cercetează "posibilitățile" unei aplicații (de exemplu, aplicația românească de tip agenda *MyCount* difuzată pe Internet): citește explicațiile asociate butoanelor, opțiunile din meniuri, informațiile din Help, experimentează operațiile propuse de aplicație imaginând utilitatea practică a acestora și formulează concluzii generale referitoare la acel soft (utilitate, spațiu ocupat pe disc sau în memorie, spațiu ocupat de produsele construite cu ajutorul lui, calitatea grafică și funcțională a interfeței). Investigării mai rafinate pot realiza elevii inițiați în programare și în sisteme de operare care disting mult mai multe aspecte "în spatele" produsului soft investigat (eficiență, posibilitate de dezvoltare, configurare, conflicte cu alte programe etc.).

Pentru organizarea activităților de investigație, profesorul va urmări:

- formularea generală a temei;
- asigurarea surselor bibliografice sau tehnice necesare;
- formularea unor indicații care să direcționeze activitatea elevilor;
- urmărirea activității elevului în sensul utilizării eficiente și creațoare a materialului de investigație;
- sprijinirea elevilor sau grupurilor de elevi care întâmpină dificultăți în înțelegerea temei sau a metodelor specifice de studiu;
- încurajarea și evidențierea activităților creațoare desfășurate de elevi, a descoperirilor neașteptate.

2. Referatul și proiectul

Referatul reprezintă o formă de îmbinare a studiului individual cu activitate de prezentare și argumentare. Tema referatului, însotită de bibliografie și alte surse de documentare (Internet, vizite etc.), este tratată în mod independent de către elev și susținută apoi în fața colegilor sau altui auditoriu mai larg. Varietatea universului informatic (a limbajelor și tehnicilor de programare, a aplicațiilor din domeniul TIC, a nouătilor hardware etc.) justifică utilizarea acestei forme de studiu și evaluare la clasă, atât la Tehnologia informației cât și la Informatică. Dacă studiul aferent și rezultatul studiului prezintă interes și din punct de vedere practic, rezultatul fiind un program (o aplicație) sau un produs TI complex (rezultatul aplicării creațoare a instrumentelor informatici), dacă bibliografia propusă este mai bogată și etapele de proiectare (concepție), implementare și testare necesită un timp mai îndelungat, lucrarea poartă numele de **proiect**.

Organizarea unei activități de evaluare și învățare prin intermediul referatelor și proiectelor presupune:

- valorificarea metodei de învățare prin descoperire;
- studiul unor materiale suplimentare și izvoare de informare diverse în scopul îmbogățirii și activizării cunoștințelor din domeniul studiat sau domenii conexe, prin completări de conținut ale programei sau prin aducerea în atenție a unei problematici complet noi;
- structurarea informației corespunzătoare unui referat într-un material ce poate fi scris, ilustrat sau prezentat pe calculator; activitățile de concepere, organizare, experimentare, reproiectare (dacă este cazul), dezvoltare și elaborare a documentației aferente necesită planificarea unor etape de elaborare și o strategie de lucru, în cazul proiectului;
- prezentarea referatului sau proiectului de către elevul sau elevii care l-au elaborat, acesta (sau un reprezentant al grupului) trebuind să-l susțină, să fie capabil să dea explicații suplimentare, să răspundă la întrebări etc.

Referatul este de regulă o lucrare de mai mică amplitudine, dar mai structurată și mai bogată în informații decât o temă de muncă independentă aferentă lecției curente. Proiectul este o lucrare mai amplă a

cărei temă este comunicată sau aleasă din timp, elaborarea unui proiect putând să dureze de la 1-2 săptămâni până la 2-3 luni sau chiar un semestru. Proiectul poate fi elaborat în grup, cu o distribuire judicioasă a sarcinilor între membrii grupului.

Referatul poate fi utilizat la Informatică și în egală măsură la Tehnologia informației temele referatelor vizând cel mai des domenii de actualitate în producția software sau în domeniul TIC.

Pentru a realiza o evaluare pe bază de referate, profesorul:

- va formula teme clare, de complexitate medie, precizând pe cât posibil amprenta lucrării (câte pagini, durata maximă necesară prezentării etc.)
- va recomanda sau asigura sursele bibliografice și de informare necesare;
- își va rezerva suficient timp (în perioada de evaluare sau la sfârșitul unor unități de învățare) pentru ca elevii să însarcinați cu elaborarea referatelor să-și poată prezenta referatul;
- va supraveghea discuțiile purtate cu elevii asupra conținutului referatului.

Pentru a realiza o evaluare pe bază de proiecte, profesorul:

- va formula teme practice, de complexitate sporită, lăsând celor care elaborează proiectul multă libertate în a improviza, adapta și interpreta cerința într-un mod personal;
- va stabili un termen final și, în funcție de modul de evaluare, termene intermediare de raportare;
- va recomanda sau asigura sursele bibliografice și de informare necesare;
- își va rezerva suficient timp (în perioada de evaluare sau la sfârșitul unor unități de învățare) pentru ca elevii să însarcinați cu elaborarea proiectelor să-și poată prezenta rezultatul proiectării;
- va supraveghea discuțiile purtate cu elevii asupra proiectului.

3. Portofoliul

Portofoliul reprezintă o metodă complexă de evaluare în care un rezultat al evaluării este elaborat pe baza aplicării unui ansamblu variat de probe și instrumente de evaluare.

Prin multitudinea de forme și momente în care se desfășoară testarea elevului, rezultatul final “converge” către valoarea reală a acestuia, sesizând elementele de progres sau regres, ilustrând preocuparea pentru lămurirea neclarităților, oferind o imagine de ansamblu asupra nivelului cunoștințelor, gradului de formare a abilităților și gradului de raportare atitudinală pe care acesta o are față de tema evaluată. Portofoliul este realizat pe o perioadă mai îndelungată, de la un semestru, un an, până la un ciclu de învățământ.

Conținutul unui portofoliu este reprezentat de rezultatele la: lucrări scrise sau practice, teme pentru casă, investigații, referate și proiecte, observarea sistematică la clasă, autoevaluarea elevului, chestionare de atitudini etc. La Tehnologia informației portofoliul se poate constitui dintr-o colecție de lucrări practice realizate pe calculator, fiecare vizând anumite aspecte de utilizare.

Alegerea elementelor ce formează portofoliul este realizată de către profesor (astfel încât acestea să ofere informații concluzive privind pregătirea, evoluția, atitudinea elevului) sau chiar de către elev (pe considerente de performanță, preferințe etc.)

Structurarea evaluării sub forma de portofoliu se dovedește deosebit de utilă, atât pentru profesor, cât și pentru elev sau părinții acestuia.

Pentru a realiza o evaluare pe bază de portofoliu, profesorul:

- va comunica elevilor intenția de a realiza un portofoliu, adaptând instrumentele de evaluare ce constituie “centrul de greutate” ale portofoliului la specificul disciplinei;
- va alege componente ce formează portofoliul, dând și elevului posibilitatea de a adăuga piese pe care le consideră relevante pentru activitatea sa;
- va evalua separat fiecare piesă a portofoliului în momentul realizării ei, dar va asigura și un sistem de criterii pe baza cărora să realizeze evaluarea globală și finală a portofoliului;
- va pune în evidență evoluția elevului, particularitățile de exprimare și de raportare a acestuia la aria vizată;
- va integra rezultatul evaluării portofoliului în sistemul general de notare.

4. Observarea sistematică a activității și comportamentului elevilor

Fișa de observare a activității și comportamentului elevului înregistrează informații legate de particularitățile personalității elevului manifestate în procesul didactic, de achizițiile evaluate spontan (răspunsuri sporadice, atitudini semnificative etc.), de progresul înregistrat de acesta. Profesorul construiește această fișă în vederea individualizării procesului sumativ de evaluare, dar și a celui de învățare.

Pe baza fișei de evaluare se poate realiza și orientarea școlară și profesională a elevului. Informațiile din fișă personală au caracter parțial secret, parte dintre ele fiind comunicate elevului și părinților acestuia.

Un model orientativ de fișă de observare conține:

- Date generale despre elev (nume, prenume, vârstă, climat educativ, condiții materiale, particularități socio-comportamentale);
- Particularități ale proceselor intelectuale (gîndire, limbaj, imaginație, memorie, atenție, spirit de observație etc.);
- Aptitudini și interese manifestate;
- Particularități afectiv-motivaționale;
- Trăsături de temperament;
- Atitudini și relaționare (cu sine însuși, cu materia studiată, cu colegii)
- Considerații privind evoluția aptitudinilor, atitudinilor, intereselor și nivelului de integrare.

Prin stabilirea competențelor generale ale disciplinei și achizițiile cognitive și comportamentale vizate de aceasta, fișa de observare poate să conțină și considerente legate de atingerea și formarea competențelor specifice. Completarea fișei se realizează în timp într-un ritm adecvat specificului activităților de la disciplina, din anul și de la clasa respectivă, dar și în funcție de implicarea și de ritmul individual al elevului.

IV. . Proiectul unității de învățare - Proiectul de lecție ?

Față de proiectarea tradițională centrată pe lecție (ora de curs) - proiectarea unității de învățare are urmatoarele avantaje:

- creează un mediu de învățare coerent în care așteptările elevilor devin clare pe termen mediu și lung;
- implică elevii în „proiecte de învățare personale” pe termen mediu și lung - rezolvare de probleme complexe, luare de decizii complexe, cu accent pe explorare și reflecție;
- implică profesorul într-un „proiect didactic” pe termen mediu și lung, cu accent pe ritmurile de învățare proprii ale elevilor;
- dă perspectiva lecțiilor, conferind acestora o structură specifică, în funcție de secvența unității de învățare în care se află.

Proiectul de lecție - conceput ca document separat - este recunoscut ca o formalitate consumatoare de timp și energie. Proiectul unei unități de învățare conține suficiente elemente pentru a oferi o imagine asupra fiecărei ore. Ca urmare, în tabelul care sintetizează proiectarea unitatii de învățare se pot delimita prin linii orizontale (punctate) spațiile corespunzatoare unei ore de curs. Astfel, pentru fiecare lecție, proiectul unității de învățare oferă date referitoare la elementele de conținut și competențele vizate, la care se raportează anumite activități de învățare; totodată, sunt indicate resurse materiale, forme de organizare a clasei etc., pentru fiecare activitate precum și instrumente de evaluare necesare la nivelul lecției (orei). În consecință, dacă proiectul unității de învățare este bine construit, nu mai este necesară detalierea la nivelul proiectului de lecție.

Lecția este înțeleasă ca o componentă operațională (**Cum?**) pe termen scurt a unității de învățare. Dacă unitatea de învățare oferă înțelegerea procesului din perspectivă strategică, lecția oferă înțelegerea procesului din perspectivă operativă, tactică. Proiectul unității de învățare trebuie să ofere o derivare simplă a lecțiilor componente. Ca urmare, trecerea de la unitatea de învățare - o entitate supraordonată - la o lecție componentă trebuie să permită o „replicare” în același timp funcțională (**De ce?**), structurală (**Cu ce?**) și operațională (**Cum?**) a unității de învățare, la o scară temporală mai mică și într-un mod subordonat.

Acest mod de tratare orientată către scopuri precise caracterizează organizarea atât a unității de învățare cât și a lecției.

Planificare calendaristică
Anul scolar 2007-2008

Nr. Crt.	Unități de învățare	Competențe specifice	Conținuturi	Nr. Ore	Săptămâna	Observații
1.	Limbajul C#	1.3, 1.4, 3.1, 3.2	<ul style="list-style-type: none"> ❖ Limbajul C# (structura unui proiect, declarări de date, citiri/scrieri folosind dispozitive standard, instrucțiuni, structuri de control, structuri de date) ❖ Interfața Visual Studio .Net, C# (meniuuri, bare de instrumente, etapele realizării unei aplicații consolă, tehnici de depanare și autodocumentare) ❖ Programare procedurală (subprograme, tehnici de transfer al parametrilor) 	12	S1-S3	
2.	Programare Orientată pe Obiecte	1.1, 1.2, 1.3, 1.4, 3.1	<ul style="list-style-type: none"> ❖ Principiile programării orientate pe obiecte ❖ Clase (structura clasei: date, metode și proprietăți, constructori, destrutor) ❖ Obiecte (instantiere, specificatori de acces, membri statici ai unei clase) ❖ Derivare (moștenire, definire și redefinire de metode, clase și metode abstracte, clase și metode sigilate, polimorfism) ❖ Ierarhii de clase predefinite (using) ❖ Aplicații consolă cu clase 	21	S4-S10	

Nr. Crt.	Unități de învățare	Competențe specifice	Conținuturi	Nr. Ore	Săptămâna	Observații
3.	Programare vizuală	1.1, 2.1, 2.2, 3.1,3.2	<ul style="list-style-type: none"> ❖ Elemente de bază ale programării vizuale (clase predefinite, interfață, evenimente, exceptii) ❖ Ferestre (crearea și particularizarea formularelor) ❖ Controle Standard, Container și Components (proprietați, evenimente specifice, metode asociate evenimentelor, Sender) ❖ Obiecte de tip dialog și obiecte grafice ❖ Tratarea exceptiilor (ierarhia claselor de erori, try-catch-finally-throw) ❖ Construirea unei aplicații vizuale (proiect, spațiu de nume, bare de instrumente, moduri de vizualizare și instrumente de depanare) 	21	S11-S17	
4.	Structuri de date în mediu vizual	1.1, 1.2, 1.3, 1.4, 3.1,3.2	<ul style="list-style-type: none"> ❖ Structuri de date înlățuitoare (liste, stive, cozi, clase generice; for each) ❖ Baze de date (conexiuni, tabele, relații, controale de tip DataSet, DataGridView, TableAdapter) ❖ Operații cu baze de date (comenzi SQL, generarea și executarea comenziilor) ❖ Construirea unei aplicații cu liste și tabele 	18	S18-S23	
5.	Dezvoltarea și prezentarea unei aplicații vizuale	2.1, 2.2, 2.3, 3.1, 3.2, 3.3	<ul style="list-style-type: none"> ❖ Ciclul de viață al unui produs software (alegerea temei, definirea cerințelor utilizator, modelarea soluției, implementarea și testarea produsului) ❖ Modele arhitecturale ❖ Realizarea și documentarea proiectului ❖ Prezentarea proiectului 	30	S24-S33	

Proiectul unității de învățare

An școlar 2007-2008

Unitatea 1. Limbajul C# (12 ore)

Nr. Crt.	Conținuturi	C.S.	Activități de învățare	Resurse	Evaluare	Observații
1.	Limbajul C# - structura unui proiect - declarări de date - citiri/scrieri folosind dispozitive standard - instrucțiuni și structuri de control - structuri de date	2.1 2.2 1.4	<ul style="list-style-type: none"> Familiarizare/repetare sintaxă C Program C# pentru o prelucrare simplă (simplificare fracție, termeni Fibonacci din interval, cifra de control etc.) Realizare aplicație consolă corespunzătoare Tipuri de date noi în C#. Siruri de caractere 	Conversația, modelarea, exercițiul Act.frontală	Observare sistematică	se preferă algoritmi din tematica de atestat și bacalaureat
2.	Interfața Visual Studio .Net (C#) - meniu și bare de instrumente - etapele realizării unei aplicații consolă - tehnici de depanare și autodocumentare	3.1	<ul style="list-style-type: none"> Discussarea structurii unui proiect simplu și enumerarea etapelor de realizare a acestuia Exersarea principalelor opțiuni din meniu, butoane și shorcururi pentru realizarea unei aplicații consolă Program de prelucrare stringuri (inversare nume-prenume, criptări sau codificări etc.) Aplicație consolă cu tablouri (ordonare, diagonale matrice etc.) 	Exercițiul, demonstrarea Act. frontală și individuală Fisa de activ. practică	Observare sistematică Investigație	
3.	Programare procedurală - subprograme - tehnici de transfer al parametrilor	2.1	<ul style="list-style-type: none"> Definire funcție (prim, palindrom etc.) și subprogram procedural (transformare, citire, afișare matrice, ordonare vector etc.) Aplicație consolă care apelează funcția și procedura definite 	Exercițiul Act.frontală și individuală Test evaluare	Fișă de evaluare 1.1 3 ore	Fișă de evaluare 1.1
4.	Realizarea unor aplicații consolă cu algoritmi elementari, structuri de date și subprograme	3.1 3.2	<ul style="list-style-type: none"> Realizarea unei aplicații-joc simple (centrate și necentrate, X și O, corăbii etc.) Discussarea algoritmilor de prelucrare, implementarea și testarea subprogramelor, implementarea aplicației consolă 	Conversația, exercițiul Act.frontală și individuală	Fișă de evaluare 1.2 3 ore	Fișă de evaluare 1.2

FISĂ DE EVALUARE 1.1

Nr.1

1. (1p.) Care dintre următoarele variante reprezintă antetul corect al unui subprogram **cif1** care primește o valoare reală prin intermediul parametrului **r** și returnează prin intermediul parametrului **z** numărul de cifre aflate la partea zecimală a lui **r** (de exemplu pentru **r=5.12703**, **z** va returna valoarea 1, iar pentru **r=126**, **z** va returna valoarea 3)

- a) static void cif1(float r, int z)
- b) static void cif1(float r, ref int z)
- c) int void cif1(float r)
- d) static float cif1(int z)

2. (1p.) Se știe că sunt declarate o variabilă **x** și ea reține o valoare reală oarecare și o variabilă întreagă **k** și ea reține valoarea 0. Care dintre următoarele variante reprezintă un apel corect al subprogramului **cif1** de la cerința 1?

- a) cif1(x,k)
- b) k=cif1(x,0)
- c) cif1(x,ref k)
- d) cif1(ref x, k)

3. (3p.) Scrieți secvența de program ce realizează prelucrarea cerută la 1. Valoarea variabilei reale **x** este considerată cunoscută, iar valoarea calculată a lui **z** nu se va afișa.

4. (3p.) Considerând implementat subprogramul **cif1**, scrieți un program care citește de la tastatură o valoare reală și, folosind apeluri ale subprogramului **cif1**, verifică dacă această valoare are mai multe cifre la partea întreagă decât la cea zecimală. Se va afișa unul dintre mesajele **DA** sau **NU**.

FISĂ DE EVALUARE 1.1

Nr.2

1. (1p.) Care dintre următoarele variante reprezintă antetul corect al unui subprogram **cif2** care are un singur parametru **r** prin intermediul căruia primește o valoare reală? **cif2** va returna numărul de cifre aflate la partea zecimală semnificativă a lui **r** (de exemplu pentru **r=5.120700**, subprogramul va returna valoarea 4, iar pentru **r=126**, va returna valoarea 0)

- a) static void cif2(ref float r, int z)
 - b) static void cif2(float r, ref int z)
 - c) static int cif2(float r)
 - d) static float cif2(ref float r)
- 2.** (1p.) Se știe că sunt declarate o variabilă **x** și ea reține o valoare reală oarecare și o variabilă întreagă **k** și ea reține valoarea 0. Care dintre următoarele variante reprezintă un apel corect al subprogramului **cif2** de la cerința 1?

- a) cif2(x)
- b) k=cif2(x)
- c) cif2(x,ref k)
- d) k=cif2(x,ref k)

3. (3p.) Scrieți secvența de program ce realizează prelucrarea cerută la 1. Valoarea variabilei reale **x** este considerată cunoscută, iar valoarea determinată se va afișa.

4. (3p.) Considerând implementat subprogramul **cif2**, scrieți un program care citește de la tastatură o valoare reală și, folosind apeluri ale subprogramului **cif2**, verifică dacă această valoare are mai multe cifre la partea întreagă decât la cea zecimală. Se va afișa unul dintre mesajele **DA** sau **NU**.

A) Planificare calendaristica
 Unitatea scolară.....
 Disciplina **INFORMATICĂ**

Professor.....
 Clasa/Nr.Ore/săpt **3 ore/săpt**

Planificarea calendaristică
 Anul școlar **2007-2008**

MODUL Programare orientată pe obiecte și programare vizuală (1 ora teorie+2 ore activități practice)

Nr. crt.	Unități de învățare	Competențe specifice	Conținuturi	Nr. ore	Săptămână	Observații
1.	PROGRAMAREA ORIENTATĂ PE OBIECTE	C1) Însușirea conceptelor de abstracțizare a datelor specific PE C2) Folosirea terminologiei variate de aplicare C3) Organizarea datelor ce intervin în rezolvarea unei probleme utilizând structuri de date adecvate C4) Utilizarea facilităților oferite de POO C5) Elaborarea și realizarea unei aplicații, folosind un mediu de programare specific	1.1.Principiile programării orientate pe obiecte 1.2.Structura unei aplicații orientată pe obiecte 1.3.Clașe și obiecte 1.4.Derivarea claselor 1.5.Tratarea erorilor 1.6.Polimorfism 17. Recapitulare	3 12 33 6 3 3 3	S1 S2-S5 S6-S16 S17-S18 S19 S20 S21	
2.	PROGRAMAREA VIZUALĂ	C6) Exprimarea și redactarea coerentă în limbaj formal sau în limbaj cotidian, a rezolvării sau a strategiilor de rezolvare a unei probleme C7) Analiza de situații-problemă în scopul descoperirii de strategii pentru optimizarea soluțiilor C8) Generalizarea unor proprietăți prin modificarea contextului inițial de definire a problemei sau prin generalizarea algoritmilor C9) Întelegerea principiilor de proiectare a interfetelor C10) Utilizarea instrumentelor de dezvoltare a unei aplicații C11) Elaborarea și realizarea unei aplicații, folosind un mediu de programare specific C12) Prezentarea unei aplicații	2.1.Concepte de bază ale programării vizuale 2.2.Prezentarea unui mediu de programare vizual (Microsoft Visual C#, Delphi, Microsoft Visual Basic etc.). 2.3. Elementele POO în context vizual 2.4.Construirea interfeței utilizator 2.5.Acesarea și prelucrarea datelor 2.6. Dezvoltarea și prezentarea unei aplicații în mediu vizual	3 3 3 3 15 12 6	S22 S23 S24 S25-S29 S30-S33 S34-S35	

B) Proiectul unitatii de invatare

Unitatea scolara.....
Disciplina...INFORMATICA
Profesor.....

Saptamana/ Anul.....
Clasa/Nr. Ore /sapt..... 3ore/săptămână

Proiectul unitatii de invatare
Unitatea de invatare PROGRAMAREA ORIENTATĂ PE OBIECTE
Nr. Ore alocate 63

Nr. Crt	Conținuturi	Competențe specifice	Activitati de invatare	Resurse		Evaluare Tip/ instrument	Obs
				Temperiale (ore)	Forme de organizare		
1.	Principiile programării orientate pe obiecte	C1	-pentru a ilustra principiile programării orientată pe obiecte și modul de structurare a unei aplicatii POO se va prezenta și analiza o aplicație gata implementată; se va realiza o analiză comparativă între abordarea POO în raport cu abordarea specifică programării structurate;	3	Frontală	Anexa 1	Initială/ Studiu de caz Jurnal Reflexiv Anexa 2
2.	Structura unei aplicatii orientată pe obiecte	C1 C2 C4	2.1.Arhitectura platformei Microsoft .NET, componente ale lui .NET Framework, trăsături ale platformei .NET 2.2.Structura generală a unei aplicații C# 2.3.Tipuri de date (predefinite, valoare, enumerare) 2.4.Tablouri	-analiza arhitecturii platformei .NET -formularea trasăturilor platformei .NET -identificarea caracteristicilor fiecărei componente ale platformei .NET -exerciții de descriere a metodei de rezolvare a unei probleme din perspectiva structurării datelor; - exerciții de utilizare a selecției, iterației și saltului - exerciții de identificare a tipurilor de date necesare în aplicații	3 Individuală Frontală Pe grupe	Anexa 3 R.A.I. Anexa 4 Anexa 5	Formativă/ R.A.I. Anexa 4 Anexa 5

	Tablouri multidimensionale		- formularea unor probleme simple; discuții preliminare și analiza problemelor;				
2.5.String-uri	C2 C4 C5		- formularea și rezolvarea unor probleme care necesită utilizarea datelor structurate	3	Individuală Frontală Pe grupe	Anexa 8	Formativă/ Exercițiul
3.	Clase și obiecte						
	3.1.Definiția claselor și a obiectelor	C2 C4	-pentru ca elevii să înțeleagă diferențele specifice abordării POO, se va alege una dintre aplicațiile familiare elevilor (de exemplu: numere complexe, numere rationale, polinoame, liste, etc.), care va fi analizată din perspectiva POO, apoi se vor implementa clasele necesare și se va realiza o aplicație în care vor fi utilizate clasele create;	3	Frontală	Anexa 9	Initială/ Exercițiul
	3.2.Utilizarea claselor și a obiectelor	C2 C3 C4	-exerciții care implică boxing și unboxing -exerciții de definire și apelare a metodelor -exerciții de utilizare a delegărilor	3	Frontală Individuală	Anexa 10 Anexa 11 Anexa 12	Formativă/ Exercițiul T 3-2-1
	3.3.Specificatorii de acces la membrii unei clase	C2 C5		6	Frontală Individuală Pe grupe	Anexa 13	Formativă/ Exerciții
	3.4.Functii. Transmiterea parametrilor	C2 C5		3	Frontală Individuală	Anexa 14	Formativă/ R.A.I.
	3.5.Crearea obiectelor; Constructori	C2 C4		3	Frontală Individuală	Anexa 15	Formativă/ Exercițiul
	3.6. Distrugerea obiectelor Destructori	C2 C4 C5		3	Frontală Individuală Pe grupe	Anexa 16	Formativă/ Exercițiul
	3.7.Operatori	C2 C3		3	Frontală Individuală	Anexa 17	Formativă/ Exercițiul
	3.8.Delegații	C3 C4		3	Frontală Individuală	Anexa 18	Formativă/ Exerciții
	3.9.Evenimente	C2 C5		6	Frontală Individuală	Anexa 19	Formativă/ Studiul de caz
4	Derivarea claselor	C2 Definirea noțiunii de moștenire simplă și moștenire multiplă Declarația unei clase derivate Drepturi de acces în clase derivate Constructori și destructor în clase	-se va analiza un exemplu de ierarhie de clase gata implementată, pentru a înțelege mecanismele care guvernează derivarea claselor;	6	Frontală Individuală Pe grupe	Anexa 21	Formativă T 3-2-1 Studiul de caz Exercițiul

5	derivate					
	Tratarea erorilor Ce este o eroare Cum se generează Tratarea/propagarea erorilor Ierarhia claselor de erori	C2 C3 C4 C5	-discuții privind validitatea datelor; -testarea și analizarea comportamentului aplicatiilor pentru diferite date de intrare;	3	Frontală Individuală	Anexa 22 Formativă/ Exerciții R.A.I.
6	Polimorfism Functii virtuale Clase abstracte și funcții virtuale pure	C2 C3 C4 C5		3	Frontală Individuală	Anexa 23 Formativă/ T 3-2-1
7	Recapitulare	C1 C2 C3 C4 C5	- fiecare elev va alege o problemă concretă, va defini clasele/ierarhile de clase necesare și va crea o aplicație în care să fie utilizate clasele/ierarhile de clase create; în acest scop se poate folosi un mediul vizual; elevii vor prezenta colegilor aplicațiile create; profesorul va stimula discuțiile critice și creative pe marginea aplicațiilor elaborate de elevi;	3	Frontală Individuală Pe grupe	Sumativă/ Project

A) Planificare calendaristică
PROGRAMARE WEB

Unitatea scolară.....
Disciplina **INFORMATICA**

Professor.....
Clasa/Nr.Ore/sapt **3 ore/săpt**

Planificarea calendaristică
Anul școlar **2007-2008**

Nr. Crt.	Unități de învățare	Competențe specifice	Conținuturi	Nr. Ore	Săpt.
1.	<i>Principii generale ale proiectării interfețelor Web</i>	1.1 Analizarea unei probleme în scopul identificării și clasificării resurselor necesare	1.Etapele procesului de dezvoltare a unei aplicații Web	2h	1
		1.1 Analizarea unei probleme în scopul identificării și clasificării resurselor necesare	2.Aspecete generale ale proiectării interfețelor Web	2h	1-2
		2.1 Identificare tehnicii de programare adecvate rezolvării unei probleme și aplicarea creativă a acestora	3. Realizare interfețelor Web utilizând HTML	7h	2-4

A) Planificare calendaristică
PROGRAMARE WEB

	3.28 Utilizarea folor de stil, a principiilor de bază ale structurii unei foi de stil 3.29. Folosirea comenzilor asociate foilor de stil CSS, ale stilurilor CSS în paginile HTML	4. Foi de stiluri (CSS) 5. Evaluare sumativă	6h 1h	4-6 6
2.	3.1 Utilizarea instrumentelor de dezvoltare a unei aplicații 3.1 Utilizarea instrumentelor de dezvoltare a unei aplicații <i>Mediu de lucru</i>	1. Modelul client - server, protocoale de comunicații 2. Server Web - IIS	1h 1h	7 7
3.	3.1 Utilizarea instrumentelor de dezvoltare a unei aplicații <i>Interacțiunea cu baze de date Web</i>	4. Prezentarea limbajului de scripting - ASP 5. Evaluare sumativă	13h 15h	7-11 12-16 17
	3.24 Operarea cu baze de date	1. Definirea și gestionarea unei baze de date	3h	18

A) Planificare calendaristică
PROGRAMARE WEB

	<p>3.25 Utilizarea informațiilor dintr-o bază de date 3.26 Aplicarea operațiilor elementare pe o bază de date</p> <p>3.1 Utilizarea instrumentelor de dezvoltare a unei aplicații 3.2 Elaborarea și realizarea unei aplicații folosind un mediu de programare specific 2.3 Analiza comparativă a eficienței diferitelor tehnici de rezolvare a problemei respective și alegerea celei mai eficiente variante</p>	<p>2. Lucrul cu baze de date</p> <p>18h</p>	<p>19-24</p>
	<p>TOTAL</p>		<p>105h</p>

Introducere în Programarea .Net Framework

Năvodari, 20-30 august 2007

www.microsoft.com/romania/educatie
<http://msdn2.microsoft.com/en-us/express/default.aspx>