

1. NOTIUNI INTRODUCTIVE. VARIABILE. OPERATORI

| | |
|--|-----------|
| 1.1. Java, primul contact..... | <u>2</u> |
| 1.1.1. Caracteristici generale..... | <u>2</u> |
| 1.1.2. Resurse necesare pentru a dezvolta programe Java..... | <u>2</u> |
| 1.1.2.1. Platforme Java disponibile..... | <u>2</u> |
| 1.1.2.2. Software necesar..... | <u>2</u> |
| 1.1.3. Pasii de realizare si rulare a unui program Java..... | <u>3</u> |
| 1.1.3.1. Imaginea de ansamblu: un exemplu..... | <u>3</u> |
| 1.1.3.2. Ce contine fisierul sursa?..... | <u>3</u> |
| 1.1.3.3. Etape de lucru si descrierea acestora..... | <u>4</u> |
| 1.2. Variabile si tipul lor de date..... | <u>5</u> |
| 1.2.1. Concepte..... | <u>5</u> |
| 1.2.2. Declararea unei variabile..... | <u>5</u> |
| 1.2.3. Initializarea unei variabile..... | <u>6</u> |
| 1.2.4. Tipuri de date in Java..... | <u>6</u> |
| 1.2.4.1. Categorii majore de tipuri de date si efectul acestora asupra variabilelor..... | <u>6</u> |
| 1.2.4.2. Tipuri de date primitive..... | <u>6</u> |
| 1.2.5. Modalitati de reprezentare a valorilor in codul sursa..... | <u>7</u> |
| 1.3. Afisarea pe ecran..... | <u>8</u> |
| 1.4. Operatori..... | <u>9</u> |
| 1.4.1. Concepte..... | <u>9</u> |
| 1.4.2. Operatorul de atribuire "="..... | <u>9</u> |
| 1.4.3. Operatorul de conversie explicita "(")..... | <u>9</u> |
| 1.4.4. Operatori aritmetici..... | <u>10</u> |
| 1.4.4.1. Operatori aritmetici binari | <u>10</u> |
| 1.4.4.2. Operatori aritmetici unari..... | <u>11</u> |
| 1.4.5. Operatori la nivel de bit..... | <u>11</u> |
| 1.4.6. Operatori de comparare..... | <u>11</u> |
| 1.4.7. Operatori logici..... | <u>12</u> |
| 1.4.8. Operatorul ternar conditional..... | <u>13</u> |
| 1.5. BIBLIOGRAFIE..... | <u>13</u> |
| 1.6. Anexa: Ghidul studentului InfoAcademy – primii pasi..... | <u>14</u> |

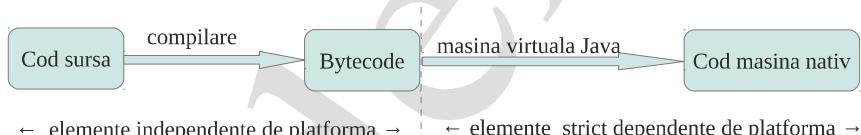
1.1. Java, primul contact

1.1.1. Caracteristici generale

Ce este Java? Java este un limbaj de programare de nivel înalt, dezvoltat în anul 1995 de către SUN Microsystems, și utilizat astăzi pe scară largă atât în aplicațiile desktop sau internet ca și pentru cele destinate dispozitivelor mobile.

De ce Java? Java a fost de la bun început un limbaj gândit pentru ușurință și eficiență creației de software, eliminând neajunsurile altor limbi (de ex. C++). Calitatea sa principală este independența de platformă – un program scris o dată poate fi rulat pe diverse arhitecturi (Intel x86, SPARC etc) fără să fie necesara ajustarea lui.

Unde este secretul? Dintotdeauna, una din problemele majore ale limbajelor a fost portabilitatea. Java rezolvă această problemă prin faptul că programele, odată compilate, generează cod masina care nu este destinat direct procesorului calculatorului gazdă, ci unui procesor virtual Java, simulație de către un software ce poartă numele de Java Virtual Machine (JVM). JVM „traduce” codul masina Java în instrucțiuni intelese de către procesorul gazdă. Existând câte o variantă JVM pentru toate sistemele de operare și arhitecturile de procesor – însă toate simulând același procesor virtual Java! – codul masina Java este practic independent de platformă și arhitectură.



1.1.2. Resurse necesare pentru a dezvolta programe Java

1.1.2.1. Platforme Java disponibile

Date fiind directiile multiple de utilizare Java și necesitatile lor diferite, Oracle oferă la momentul scrierii acestui material trei platforme Java:

- **Java SE** (Standard Edition) – platforma Java de bază pentru aplicații desktop sau server. Cuprinde componentele necesare pentru orice aplicație Java, inclusiv design de interfețe grafice complexe
- **Java EE** (Enterprise Edition) – platforma Java dedicată aplicațiilor server, cu grad sporit de complexitate. Java EE include resursele disponibile în Java SE și adaugă elemente necesare serverelor de aplicație folosite pentru aplicații distribuite
- **Java ME** (Micro Edition) – platforma Java dedicată dispozitivelor portabile sau cu resurse limitate (telefoane mobile, PDA-uri etc). Platforma include facilități built-in de adaptare a platformei Java la platforma hardware pe care va rula, facilități de comunicare în rețea și de realizare de interfețe grafice etc.

1.1.2.2. Software necesar

Pentru a putea scrie și rula programe Java în mod eficient, programatorul are nevoie de următoarele resurse:

- **Java Runtime Environment (JRE)** – reprezintă totalitatea resurselor necesare pentru a putea rula programe Java. Printre ele se numără masina virtuală și un set minimal de clase predefinite Java, corespunzătoare operațiilor de întâlnire în programare (ex: lucrul cu siruri de caractere, operații matematice uzuale etc)
- **Java Development Kit (JDK)** – pachetul care conține resursele necesare creației de programe Java: compilator, generator de documentație, arhivator și o întreagă serie de alte utilități implicate direct sau indirect în procesul de dezvoltare. În general JDK conține și JRE, pentru a putea rula aplicațiile create.
- **un mediu de dezvoltare Java (IDE - Integrated Development Environment)**. Deși codul Java poate fi scris folosind orice editor de text, un mediu de dezvoltare oferă multiple facilități care sporesc eficiența programatorului (un editor avansat, managementul facil al proiectelor și resurselor atașate acestora, debugging avansat etc). Putem privi IDE-ul ca pe o interfață la uneltele puse la dispozitivul de JDK; IDE-ul se folosește de ele și nu poate funcționa în lipsa lor.

Sunt disponibile două categorii de medii de dezvoltare:

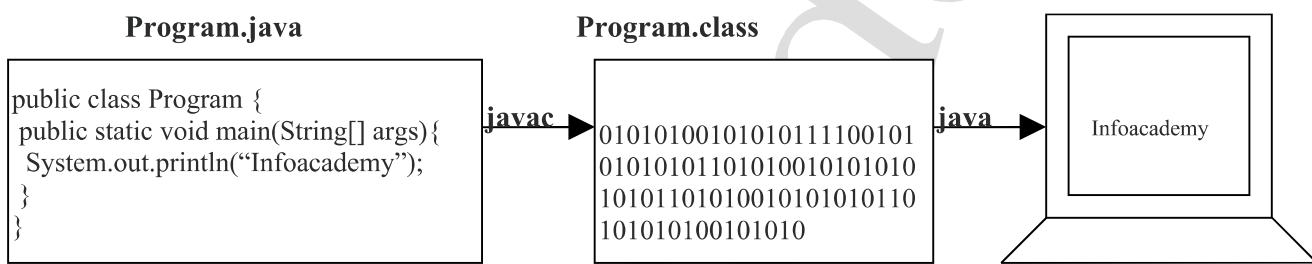
- medii de dezvoltare open-source - sunt softuri public disponibile, care pot fi descarcate si rulate de catre orice utilizator. Proeminente sunt NetBeans (care beneficiaza de sprijin din partea Sun si ulterior Oracle) si Eclipse (proiect initiat si sprijinit de IBM)
- medii de dezvoltare proprietare - sunt softuri disponibile contra cost. Exemple: JBuilder, IntelliJ Idea, MyEclipse etc.

Atat JRE cat si JDK sunt public disponibile pe <http://java.sun.com>. JDK poate fi descarcat in cel putin doua variante - de sine statator („stand-alone”) sau la pachet cu mediul de dezvoltare NetBeans („bundle”).

1.1.3. Pasii de realizare si rulare a unui program Java

1.1.3.1. Imaginea de ansamblu: un exemplu

Programele scrise in limbajul Java iau forma unui ansamblu de fisiere text cu extensia `.java`, ce contin asa-numitul *cod sursa* (instructiunile Java ce compun programul). Aceste fisiere sunt apoi compilete cu ajutorul compilatorului `javac`, rezultand un set de instructiuni in cod masina ce se va executa pe procesorul virtual Java si care poarta denumirea de *bytecode*. Bytecode-ul este plasat intr-un fisier avand același nume cu cel sursă, dar cu extensia `.class`. La rulare, masinii virtuale i se paseaza ca argument numele clasei ce reprezinta punctul de intrare al programului; masina virtuala incarca bytecode-ul clasei corespunzatoare (fisierul `.class`) si executa codul cuprins in fisier.



Nota: privind imaginea de mai sus, deducem ca Java este atat un limbaj compilat (codul sursa este transformat in bytecode) cat si unul interpretat (bytecode-ul este interpretat de catre masina virtuala Java astfel incat sa poata fi executat pe procesorul gazda).

1.1.3.2. Ce contine fisierul sursa?

Java este un limbaj de programare orientat pe obiect, si de aceea in orice fisier sursa va fi declarata o **clasa**. Dupa cum va fi explicat detaliat intr-un alt capitol, o clasa specifica structura unei categorii de obiecte – de exemplu, clasa Student descrie componitia obiectelor de tip student, urmand ca pe baza ei sa fie create efectiv aceste obiecte. Clasa reprezinta pentru obiecte ceea ce este schema electronica pentru un aparat – pe baza unei singure scheme pot fi realizate unul sau mai multe aparate.

Scheletul unui fisier sursa este prezentat mai jos (liniile care incep cu // sunt comentarii si vor fi ignorate la compilare):

```

public class PrimaClasa {
    public static void main(String[] args) {
        // aici este corpul functiei main, unde vor fi plasate instructiunile Java executate la rularea clasei
    }
}

```

In prima faza vom face abstractie (pe cat posibil) de capabilitatile obiectuale ale limbajului Java si vom plasa codul programelor noastre in functia `main()`, el fiind astfel executat automat la rularea clasei. In capitolele urmatoare vor fi introduce noi notiuni care vor clarifica pe deplin structura unei clase si modalitatatile sale de utilizare. Pe moment, este suficient sa retinem ca, **pentru a putea cod Java, este necesar sa definim o clasa iar aceasta sa contina functia main() cu semnatura de mai sus**.

1.1.3.3. Etape de lucru si descrierea acestora

Pasii parcursi in scrierea si rularea unui program Java sunt urmatorii:

1. Se editeaza fisierul/fisierele sursa si se salveaza. Un program Java contine de obicei mai multe clase; fiecare clasa este salvata in propriul sau fisier. Fisierul trebuie sa aiba numele identic cu al clasei (tinand cont de litere mici/mari!) si extensia **.java** (in cazul nostru, **Program.java**). Prin conventie, numele de clase Java incep cu litera mare iar, daca numele este format din mai multe cuvinte, fiecare cuvant incepe de asemenea cu litera mare (ex: **PrimulMeuProgram**)

Nota: un fisier sursa poate contine mai multe clase Java, insa dintre ele una singura poate avea modificarul public si aceea impune numele fisierului.

2. Se compileaza fisierul sursa, folosind compilatorul Java. Aceasta ia forma unui fisier executabil, numit **javac.exe** pe Windows sau **javac** in Unix/Linux. Spre exemplu, pentru a compila programul de mai sus, folosim linia de comanda Windows pentru a naviga pana la locul in care se afla fisierul sursa si scriem:

```
c:\java> javac Program.java
```

In cazul in care apar erori, compilatorul semnaleaza si numarul liniei/liniilor „vinovate”. Cand nu exista erori si comilarea se incheie cu succes, nu se afiseaza nimic!

Rezultatul compilarii va fi aparitia, in acelasi director, a unui fisier numit **Program.class**, ce reprezinta bytecode-ul.

3. Se ruleaza aplicatia, apeland explicit masina virtuala Java si pasandu-i ca argument numele clasei ce contine functia **main()**. Masina virtuala ia forma unui executabil denumit **java.exe** in Windows si **java** in Unix/Linux. Pentru a rula programul de mai devreme sub Windows, scriem:

```
c:\java> java Program
```

Masina virtuala primeste ca argument numele clasei ce se doreste a fi rulata. In urma comenzii anterioare, masina virtuala (executabil **java.exe**) va cauta in directorul curent un fisier cu numele **Program.class** si il va executa.

Atentie! Masina virtuala nu primeste ca argument un nume de fisier, ci unul de clasa! Nu adaugati extensia **.class** la numele fisierului cand rulati programul! Veti obtine o eroare.

La rularea unei clase se pot pasa argumente in linia de comanda; acestea vor fi disponibile prin intermediul variabilei **args**, parametrul functiei **main**. Fie clasa de mai jos:

```
class Parametri{
    public static void main(String[] args){
        System.out.println("Salut "+ args[0]+ // folosim primul argument pasat in linia de comanda
                           "!");
    }
}
```

La rulare putem pasa ca argument un nume de persoana; ruland programul de mai multe ori vom obtine output-uri diferite:

```
c:\java> java Parametri Mihai
Salut Mihai!
c:\java> java Parametri Ioana
Salut Ioana!
```

1.2. Variabile si tipul lor de date

1.2.1. Concepte

La fel ca in majoritatea limbajelor de programare, in Java folosim variabile pentru a ne referi la date (informatie) stocate in memoria calculatorului. Altfel spus, putem denumi - intr-un mod cat mai convenabil noua - valori numerice, siruri de caractere sau caractere singulare, sau chiar obiecte create de noi insine (asemanator cu modul in care lucram „in litere” in problemele de fizica, cu deosebirea ca aici variabilele nu au exclusiv valori numerice).

O variabila reprezinta o zona de memorie in care programatorul poate stoca, pe rand, diferite valori de-a lungul executiei programului. Zona de memorie in cauza este referita prin intermediul unui nume - asa-numitul *identifier* al variabilei (sau numele variabilei, in limbaj uzual).

Atentie! A nu se confunda numele variabilei cu variabila! Variabila reprezinta zona de memorie, iar numele (sau identifierul) modalitatea de a o referi in scopul citirii/modificarii valorii ei.

Orice variabila Java are un tip de date. Tipul de date al unei variabile reprezinta natura informatiei ce poate fi stocata in respectiva zona de memorie (ex: numar intreg, numar cu zecimale, sir de caractere, valoare de adevar etc.). Dupa cum se va vedea, in Java tipul de date al variabilei este stabilit cu ocazia declararii acesteia si nu se mai schimba pe parcursul existentei ei.

Rostul variabilelor in programare este sa permita programatorului lucru cu date fara a avea nevoie sa stie exact locatia datelor in momentul rularii programului. Spre exemplu, atunci cand dorim sa-i cerem procesorului sa adune doua numere, trebuie sa-i furnizam instructiunea in cod masina corespunzatoare si care isi citeste datele fie din registrele procesorului, fie din memoria RAM. In ambele cazuri, ar trebui sa stim exact unde se gasesc cele doua numere - ceea ce nici nu este eficient (programatorul trebuie sa se concentreze pe a elabora un cod functional, nu pe aspecte secundare), nici intotdeauna posibil (un acelasi program, rulat de mai multe ori, se poate gasi in zone de memorie diferite, si implicit datele cu care el opereaza vor fi stocate in locatii diferite la fiecare rulare).

1.2.2. Declararea unei variabile

Java este un limbaj de programare *strongly typed* – fiecare variabila are un tip de date bine stabilit inca de la compilare si care nu se schimba pe parcursul executiei. Ca efect, compilatorul poate verifica corecta utilizare a variabilei in cauza de-a lungul programului (compatibilitatea combinarii ei cu alte valori), prevenind astfel numeroase erori de programare.

Fiecare variabila din program trebuie *declarata* inainte de utilizare. Declararea presupune specificarea tipului de date al variabilei si a identifierului (numelui) acesteia, in aceasta ordine, separate prin unul sau mai multe spatii sau TAB-uri. Exemplu:

```
int nr;
```

Instructiunea de mai sus ii transmite compilatorului ca, atunci cand transforma programul Java in bytecode, sa aloce un spatiu de memorie de 32 de biti (dimensiunea tipului int - vezi mai jos). Compilatorul retine locatia de memorie in care este creata variabila si inlocuieste orice aparitie a identifierului *nr* in restul programului cu o accesare a acelei locatii concrete de memorie. In acest fel putem lucra cu variabila prin intermediul numelui *nr* fara a fi preocupati de adresa exacta la care este memorata informatie.

O implicatie majora a stabilirii tipului de date al variabilei este plaja de valori pe care aceasta o poate capata de-a lungul existentei. In exemplul nostru, s-au alocat 32 de biti; daca am fi folosit *byte* in loc de *int*, spatiul alocat variabilei ar fi fost de 8 biti si domeniul de valori posibile s-ar fi restrans drastic.

1.2.3. Initializarea unei variabile

O variabila poate fi initializata (poate primi o prima valoare) cu ocazia declararii, dupa cum urmeaza:

```
int nr = 4;
```

Efectul este ca in zona de memorie referita cu numele *nr* se stocheaza de la bun inceput valoarea 4.

In cazul in care variabila nu este initializata la declarare, masurile luate de compilator difera in functie de tipul variabilei:

- pentru variabile locale (cum sunt cele pe care le vom folosi in prima faza in acest curs), compilatorul nu aloca variabilei nicio valoare, insa **nu permite utilizarea acesteia neinitializata**
- pentru campuri ale clasei, variabila este automat initializata cu tipul de date default corespunzator tipului ei de date (vezi tabelul de mai jos si lectia despre clase si obiecte)

1.2.4. Tipuri de date in Java

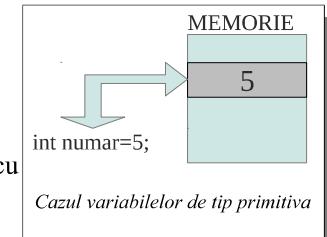
1.2.4.1. Categorii majore de tipuri de date si efectul acestora asupra variabilelor

Java defineste (si poate opera cu) urmatoarele categorii de tipuri de date:

- primitive** – tipuri de date scalare, cunoscute din oficiu de catre Java („built-in”). Locatia de memorie corespunzatoare unei variabile de tip primitiva va stoca chiar valoarea acesteia. Tipurile de date suportate sunt prezентate in tabelul de mai jos
- tipuri de date compuse** - reprezinta tipuri date in cazul caror o variabila memoreaza un grup de valori. Mentionam aici:
 - tablouri - o variabila de tip tablou contine o succesiune de valori de acelasi tip (vezi lectia despre tablouri)
 - clase - o clasa reprezinta un tip de date compus, ce poate fi creat de catre programator. Exista clase Java predefinite, iar in plus fata de ele programatorul isi defineste propriile clase

In functie de tipul lor de date, variabilele se manifesta fundamental diferit:

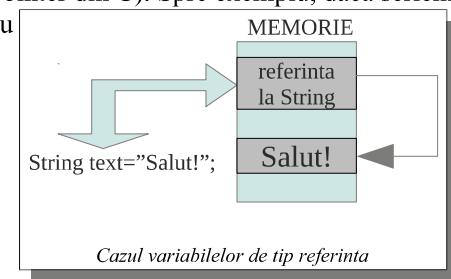
- variabilele al caror tip de date este o primitiva contin chiar valoarea in cauza. Exemplu: declaratia `int nr=5;` creeaza o zona de memorie pusa in corespondenta cu numele *nr*, si care contine chiar valoarea 5.
- variabilele cu tip de date compus (asa-numitele obiecte) nu contin direct valoarea vizata, ci o valoare de tip **referinta**. O referinta reprezinta o modalitate de a ajunge la date (conceptul este asemanator - dar nicidcum identic! - cu cel de pointer din C). Spre exemplu, daca scriem `String hobby = "alpinism";` numele *hobby* va fi pus in corespondenta cu o zona de memorie care NU contine sirul de caractere *alpinism*, ci o modalitate de ajunge la el, iar sirul *alpinism* va fi stocat intr-o zona separata de memorie.



1.2.4.2. Tipuri de date primitive

Tipurile de date primitive sunt urmatoarele:

- tipuri de date numerice
 - numere intregi - tipurile de date corespunzatoare sunt byte, short, int si long
 - numere cu zecimale - sunt valori memorate in virgula mobila. Tipuri de date corespunzatoare: float si double
- caractere - tipul de date char, care memoreaza caractere Unicode. Atentie! Un astfel de caracter este memorat in masina virtuala pe doi octeti! (spre deosebire de caracterele ASCII carora le corespunde cate un singur octet)
- valori de adevar - tipul de date boolean. Singurele valori posibile sunt true si false



| Tipul de date | Natura datelor | Nr. biti | Valori posibile | Valoare default pt campuri ale clasei | Exemplu de declarare |
|----------------|---|------------|---|---------------------------------------|---|
| byte | Numar intreg | 8 biti | -128...+127 | 0 | byte b=102; |
| short | Numar intreg | 16 biti | -32768...+32767 | 0 | short s=13; |
| int | Numar intreg | 32 biti | -2147483648..+ 2147483647 | 0 | int i=1000; |
| long | Numar intreg | 64 biti | -2 ⁶³ ...(+2 ⁶³ -1) | 0L | long l=1000000L; (L poate lipsi) |
| float | Numar rational | 32 biti | | 0.0f | float f=12.56f; |
| double | Numar rational | 64 biti | | 0.0d | double d=12.56; |
| char | Caracter Unicode | 16 biti | 0 Unicode ... 65535 Unicode | 0 sau \u0000 | char c='a'; char c='\u00a1'; |
| boolean | Valoare de adevar | irrelevant | true, false (nu sunt admise 0 si 1!) | false | boolean b=true; |
| void | folosit pentru a indica faptul ca o functie nu returneaza nimic; nu se pot declara variabile de tip void) | | | | |

ATENTIE!

- toate primitivele numerice din Java au semn! (nu exista conceptul de unsigned int, ca in C)
- tipul de date **char** este percepuit de catre compilator si masina virtuala ca un intreg fara semn si chiar poate fi folosit ca atare in expresii, in combinatie cu alte tipuri numerice (ex: suma unui char cu un int). Domeniul sau de valori este 0...65535.
- tipurile de date float&double sufera de imprecizia inerenta memorarii in virgula mobila. Ele nu trebuie folosite pentru a stoca valori exacte (ex: diverse sume de bani/solduri bancare care apoi trebuie sa produca un bilant corect). Pentru astfel de operatii exista clase Java predefinite

1.2.5. Modalitati de reprezentare a valorilor in codul sursa

In codul sursa apar deseori valori explicite de diferite naturi - asa-numitele „literals”. Spre exemplu, atunci cand initializam o variabila la declarare:

```
String s = "4";
```

Valoarea "4" este un *literal* de tip String.

Felul in care este reprezentata o valoare determina interpretarea acesteia de catre compilator ca fiind de un anumit tip de date. Compilatorul va folosi acest tip de date pentru a verifica corecta utilizare a valorii in expresia din care face parte. De aceea trebuie sa existe modalitati de reprezentare distincte pentru diversele tipuri de date; spre exemplu, putem avea numarul 4, caracterul 4 sau String-ul ce contine caracterul 4.

Regulile de reprezentare a valorilor pentru diversele tipuri de date sunt urmatoarele:

- numere
 - intregi - accepta mai multe reprezentari posibile:
 - zecimal: 14, -23
 - hexazecimal: 0xA1, -0x8C
 - binar (incepand cu Java 7): 0b110101
 - toate reprezentarile de mai sus sunt automat percepute ca int; pentru valori de tip long se poate adauga la oricare dintre ele sufixul l (L mic) sau L. Este de preferat L deoarece litera mica poate sa se confundă cu cifra 1 (unu)
 - in virgula mobila - exista doua reprezentari posibile:
 - standard
 - float - este necesara litera f la sfarsitul valorii: 1.13f, -3.178f
 - double - un numar cu zecimale fara sufixul f este considerat automat de tip double: 19.4, -51.6. Optional poate fi folosit si sufixul d sau D
 - stiintifica: 1.3e2 sau 1.3E2 sunt echivalente cu 1.3×10^2 . In functie de sufix (sau absenta acestuia), valoarea este perceputa ca fiind float sau double, conform regulilor anterioare
- caractere: valoarea trebuie cuprinsa intre apostroafe, care insa sunt o simpla modalitate de semnalizare, nu fac parte din valoare! Reprezentari posibile:
 - standard: 'B', 'z', '#', '"' (caracterul ghilimele)
 - cod Unicode: prefixul \u urmat de 4 cifre hexazecimal ce formeaza codul caracterului: '\u0041' este echivalent cu 'A'
- valori de adevar: tipul de date boolean accepta doar doua valori, **true** si **false**, scrise cu litere mici

- String: desi nu este un tip de date built-in, ci o clasa separata, String este adanc inradacinat in functionarea masinii virtuale, fiind indispensabil acestia. Valorile de tip String trebuie incadrate intre “(ghilimele). In cadrul unei valori de tip String pot fi folosite niste constructii speciale - asa-numitele sevente escape - ce corespund caracterelor de control ASCII. Exemple: \n (line feed - trecerea pe linia urmatoare), \r (carriage return - intoarcerea cursorului la inceput de rand), \t (tab) etc.

Incepand cu Java 7 a fost introdus un nou element de sintaxa conform caruia pot fi intercalate caractere _ (underscore) in cadrul valorilor numerice, pentru a imbunatatii lizibilitatea acestora. Aceasta abordare este utila pentru valori care in practica se obisnuiesc sa fie reprezentate intr-un anume format. Exemple:

```
// un numar de telefon de Bucuresti
long l = 631_75_43;
// un numar de card
long l = 0354_7840_3467_1289;
// o adresa MAC
long l = 0xC3_70_6A_16_43_BD;
```

Atentie! Caracterul _ trebuie sa fie bordat de cifre! El nu poate fi folosit la inceputul sau la sfarsitul numarului, sau in stanga sau dreapta separatorului zecimal, si cu atat mai putin in mijlocul sevantei ce stabileste baza de numeratie (ex: 0_xA1, 0_b1011).

1.3. Afisarea pe ecran

In Java, afisarea pe ecran poate fi realizata in multiple moduri, dintre care amintim deocamdata urmatoarele doua instructiuni:

```
System.out.print(expresie);
System.out.println(expresie);
```

Diferenta dintre ele este urmatoarea: varianta *println()* trece cursorul pe linia urmatoare, pe cand *print()* il mentine pe pozitia curenta astfel incat se poate adauga informatie pe acelasi rand:

```
System.out.print("Un cuvant");
System.out.print(" si inca unul");      // output pana aici: Un cuvant si inca unul
System.out.println("Linia unu");
System.out.print("Linia doi");          // output: Linia unu
                                         Linia doi
```

Expresia primita ca argument la afisare poate avea orice tip de date Java (primitiva sau nu). Ea poate fi alcatauita prin compunerea a mai multe String-uri, valori de variabile, valori de expresii:

```
System.out.println("Ma cheama " + nume + " si am " + (varsta-10) + "ani");
```

Observatii:

- una dintre primele modalitati disponibile de diagnostic al unui program este afisarea pe ecran a valorii variabilelor de interes in diverse puncte cheie ale executiei. Nu trebuie insa uitat ca mediile de dezvoltare (IDE) ofera suplimentar modalitati de debugging avansate (rulare pas cu pas, insotita de inspectia simultana a variabilelor sau expresiilor dorite)
- instructiunile System.out.print() si System.out.println() scriu in asa-numita *consola Java*. Consola reprezinta un spatiu de afisare folosit in general pentru diagnostic si nu este disponibila, spre exemplu, in cazul unei aplicatii cu interfata grafica. Vom folosi pe moment consola ca modalitate primara de interactiune cu utilizatorul, urmand ca in cadrul lectiilor despre aplicatii grafice sa adaugam si alte moduri de afisare de informatie

1.4. Operatori

1.4.1. Concepțe

Scopul principal al programării este realizarea de aplicații care lucrează cu datele. Acest lucru presupune combinarea diverselor variabile prezente în program (și de fapt, a datelor pe care acestea le denumesc) pentru a produce noi valori.

Operatorii corespund operațiilor posibile cu datele și iau forma unor simboluri (+, <, = etc). Un operator folosește una, două sau trei valori (numite operanți) pentru a produce o valoare rezultată. Spre exemplu, operatorul de adunare „+” folosește doi operanți numeric și generează o informație nouă ce reprezintă suma acestora.

Valorile pe care un operator le combina (operanții) pot fi:

- valori de variabile
- valori specificate explicit în codul sursă („literals”)
- valori returnate de funcții sau metode

Prin combinarea acestor elemente se formează *expresii*.

Operatorii pot fi clasificați din cel puțin două puncte de vedere:

- în funcție de numărul de operanți, distingem operatori unari (un operand), binari (doi operanți) și operatorul ternar (trei operanți)
- în funcție de natura operației efectuate și a rezultatului produs:
 - operatori aritmetici - realizează operații matematice uzuale (adunare, înmulțire, incrementare etc)
 - operatori de conversie - realizează „trecerea” unei valori dintr-un tip de date în altul
 - operatori logici - folosiți pentru a compune mai multe valori de adevar într-o expresie complexă
 - operatori de comparare - utilizati pentru comparare de egalitate sau inegalitate
 - operatori la nivel de bit - sunt cei care implementează operații de tip AND, OR, XOR, NOT la nivel de bit
 - operatorul ternar - utilizat ca modalitate simplă de a implementa decizii în cadrul expresiilor

1.4.2. Operatorul de atribuire “=”

Este folosit pentru atribuirea de valori variabilelor. Acceptă doi operanți – un nume de variabilă (operandul stang) și valoarea sa (cel drept). Odată declarată o variabilă x (**int x;**), ei îi se poate atribui o valoare cu instrucțiunea **x=expresie**. Atribuirile se fac după reguli bine stabilite:

- în stanga operatorului „=” se poate afla doar un nume de variabilă (primul operand nu poate fi o expresie sau o constantă: o atribuire 5=x; este interzisă)
- în dreapta operatorului „=” se poate afla o valoare, o variabilă sau o expresie, cu respectarea condiției următoare: tipul de date din dreapta trebuie să fie compatibil cu („sa incapa in”) cel din stanga; de exemplu, dacă am declarat anterior int x; nu vom putea face atribuirea x=15.7 deoarece 15.7 este o valoare de tip double.
- la întâlnirea operatorului de atribuire se evaluatează mai întâi membrul drept de la stanga la dreapta, respectând precedența operatorilor
- la atribuire pot avea loc conversii隐式 de largire de tip (ex: double d=135; deși tipurile de date ale operanților sunt diferite (cel drept este *int*), nu obținem o eroare de compilare deoarece în totdeauna un *int* "incapă" într-un *double* și compilatorul îl va promova automat la *double*)

Există și operatori de atribuire compusi, formati prin combinarea operatorului de atribuire cu un altul. Exemplu: **x+=3** realizează același lucru ca **x = x+3**; **x%=-4** înseamnă **x = x % -4** etc.

1.4.3. Operatorul de conversie explicită “()”

Este folosit pentru a converti forțat valoarea unei variabile sau expresii la un tip de date dorit. Poate fi folosit atât pentru expresii al căror tip de date este o primitivă, cât și pentru referințe, însă regulile legate de conversia referințelor vor fi acoperite într-un capitol viitor.

In cazul primitelor, nu orice conversie este permisa:

- se poate converti intre tipurile de date numerice: byte, short, int, long, float, double (si – sa nu uitam – char!)
- tipul de date boolean NU poate fi convertit din si in alte tipuri de date!

In cazul unei conversii care reduce anvergura tipului de date se aplica urmatoarele reguli:

- la conversia unui intreg intr-un alt tip de date intreg mai mic, se pastreaza bitii cei mai putin semnificativi (ex: la conversia din *int* in *byte* vor fi pastrati doar primii 8 biti)
- la conversia unui numar in virgula mobila la un tip de date intreg, se converteste intai numarul la un tip de date intreg (long sau int, in functie de magnitudine) si apoi, daca este cazul, se pastreaza doar bitii cei mai semnificativi

Atentie! In urma conversiei fortate a unui numar pozitiv poate rezulta unul negativ! (deoarece combinatia de biti ramasa dupa trunchierea numarului poate avea ca semnificatie o valoare negativa – numerele sunt reprezentate in complement fata de 2 si in aceste conditii bitul cel mai semnificativ da semnul numarului)

1.4.4. Operatori aritmetici

Sunt folositi pentru realizarea operatiilor aritmetice comune. Exista mai multe tipuri de astfel de operatori:

1.4.4.1. Operatori aritmetici binari

Operatorii aritmetici binari Java sunt: + pentru adunare, - pentru scadere, / pentru impartire, * pentru inmultire, % pentru modulo.

Este respectata ordinea operatiilor asa cum o cunoastem din matematica (* si / au prioritate fata de + si -).

Operatorii aritmetici din Java prezinta cateva particularitati importante:

- operatorul + este supraincarcat¹:
 - atunci cand operanzii sunt numere, el realizeaza adunare, si rezultatul sau va fi un numar (atentie! nu uitati ca si tipul de date char este percepuit tot ca numar!)
 - cand cel putin unul dintre operanzi este de tip String (sir de caractere), se realizeaza concatenare si rezultatul produs va fi un String
- operatorul / realizeaza impartire intreaga (produce ca rezultat catul impartirii) daca ambii operanzi sunt de tip intreg
- operatorii aritmetici (dar si cei la nivel de bit) promoveaza ambii operanzi la tipul de date cel mai larg dintre cele doua, sau cel putin la int. Tipul de date al rezultatului va fi asadar cel putin int. In functie de tipul de date al operanzilor operatorul poate furniza valori diferite (ex: in cazul impartirii, pentru operanzi intregi rezultatul este catul (intreg), insa daca macar unul dintre operanzi este float sau double rezultatul va fi float: *int x=7/2* va asigna lui x valoarea 3 de tip int, iar *int x=7.0/2* va genera o eroare de compilare deoarece operandul drept este de tip double).

Nota: in Java, programatorul nu mai poate face supraincarcare de operatori. Cativa dintre operatori sunt supraincarcati din oficiu, insa nu pot fi modificati.

Exemple:

```
System.out.println(2+3+4);           //9 (tip de date int)
System.out.println(2+3+"4");          //54 (tip de date String)
System.out.println("2"+3+4);          //234 (tip de date String)
System.out.println("2"+(3+4));        //27 (tip de date String)
System.out.println(9/2);              //4 (tip de date int)
System.out.println(9/2.0);            //4.5 (tip de date double)
```

1 Un operator supraincarcat este unul care actioneaza diferit in functie de tipul de date al operanzilor

Studentul poate utiliza prezentul material si informatiile continute in el exclusiv in scopul asimilarii cunostintelor pe care le include, fara a afecta dreptul de proprietate intelectuala detinut de InfoAcademy.

1.4.4.2. Operatori aritmetici unari

Numele lor provine din faptul ca au un singur operand. Ei sunt:

1. „++”: operatorul de incrementare (crestere cu 1). Exemplu: `x++` sau `++x`
2. „--”: operator de decrementare (scadere cu 1). Exemplu: `x--` sau `--x`

Acesti operatori pot fi plasati inaintea operandului (*prefix*) sau dupa acesta (*sufix*). Operatorul are in ambele cazuri ca efect modificarea valorii variabilei; aceasta modificare insa survine fie inainte de a fi citita valoarea variabilei (in cazul prefix) fie dupa citire. De aceea, in cazul prefix rezultatul produs este valoarea deja modificata a variabilei, pe cand in cazul sufix operatorul va produce valoarea initiala (dar variabila va fi totusi modificata in memorie imediat dupa aceea!).

Sa consideram atribuirea `y=x++ + 2`; si sa presupunem ca valoarea initiala a lui `x` este 3. Efectuarea adunarii se face prin citirea pe stiva a celor doi operanzi si apoi inlocuirea acestor doua valori cu rezultatul adunarii. Expresia din dreapta operatorului de atribuire va fi evaluata de la stanga la dreapta: intai este citita si pusa pe stiva valoarea variabilei `x` din memorie (3), apoi este incrementata valoarea din locatia de memorie (astfel ca in memoria referita de numele `x` se gaseste acum valoarea 4), iar apoi se efectueaza operatia aritmetica rezultand valoarea $3+2=5$; in final aceasta valoare va fi scrisa in locatia de memorie referita de numele `y`. Valori finale: `x=4`, `y=5`.

Asignarea `y=++x + 2`; va incrementa intai valoarea din locatia de memorie corespunzatoare lui `x`, si abia apoi il va citi pe `x` si va atribui rezultatul adunarii lui `y`. Valori finale: `y=6`, `x=4`.



Aplicatie: Incercati sa notati in coloana alaturata ceea ce va asteptati sa fie afisat pe ecran, si apoi verificati-via compiland si ruland programul.

```
public class c3 {
    public static void main(String[] args) {
        int x=1;
        System.out.println("x : "+ x);
        System.out.println("++x : "+ ++x);
        System.out.println("x++ : "+ x++);
        System.out.println("--x : "+ --x);
        System.out.println("x-- : "+ x--);
        System.out.println(" x : " + x);
    }
}
```

| Afiseaza | Val. lui x in memorie |
|----------|-----------------------|
| | |
| | |
| | |
| | |
| | |
| | |

1.4.5. Operatori la nivel de bit

Java ofera operatori pentru toate operatiile traditionale la nivel de bit: AND (`&`), OR (`|`), XOR (`^`), shift (deplasare binara) cu semn (`<<, >>`), shift fara semn (`>>>`). Numerele negative sunt reprezentate in complement fata de 2. In cazul numerelor negative shiftate la dreapta cu semn, bitii introdusi din partea stanga vor fi 1 (este pastrat bitul de semn), iar shiftarea la dreapta fara semn introduce biti zero. In cazul numerelor pozitive nu conteaza varianta de shiftare la dreapta folosita.

```
public class Deplasare {
    public static void main(String[] args) {
        int x=215;
        // in java 7: int x = 0b11010111;
        System.out.println("Bitul 0 este: "+(x&1));           // 1
        System.out.println("Bitul 1 este: "+((x>>1)&1));      // 1
        System.out.println("Bitul 2 este: "+((x>>2)&1));      // 1
        System.out.println("Bitul 3 este: "+((x>>3)&1));      // 0
    }
}
```

1.4.6. Operatori de comparare

Operatorii de comparare primesc ca operanzi doua variabile numerice si genereaza un rezultat de tip boolean in functie de relatia dintre cele doua valori. Operatorii sunt: `<`, `>`, `<=` (mai mic sau egal), `>=` (mai mare sau egal), `!=` (diferit).

ATENTIE! Din cauza impreciziei numerelor memorate in virgula mobila, nu este recomandabila compararea acestora folosind operatorul '==' ! Iata un exemplu:

```
float f = 6.1f;
double d = 6.1;
System.out.println(f);      // afiseaza 6.1
System.out.println(d);      // afiseaza 6.1
System.out.println(f==d);   // surpriza: afiseaza false!!
System.out.println(f-d);   // afiseaza -9.536743128535363E-8; valorile memorate nu sunt identice!
```

1.4.7. Operatori logici

Operatorii logici sunt cei prin intermediul caror se pot combina mai multe expresii logice individuale formand o expresie compusa, la fel ca in logica predicatorilor: SI logic (&), SAU logic (||), SAU logic exclusiv (^), NU logic (!).

Exista de asemenea operatorii logici cu scurtcircuitare: SI logic(&&) si SAU logic(||). Diferenta fata de cei anteriori este ca evaluarea operandului al doilea va avea loc numai atunci cand primul nu este suficient pentru a deduce valoarea expresiei, dupa cum urmeaza:

- in expresia v1 && v2, daca v1 este false atunci cu certitudine rezultatul expresiei este false si nu se mai evaluateaza al doilea operand
- in expresia v1 || v2, daca v1 este true atunci rezultatul intregii expresii este true si deci v2 nu mai este evaluat

Exemplu: expresia $(a>2) | (b++<5)$ se va evalua determinand intai rezultatele celor doi operanzi de comparare, ceea ce inseamna ca b va fi intotdeauna incrementat. In schimb, daca folosim operatorul scurtcircuitat, b va fi incrementat numai daca $a>2$!

```
public class C2 {
    static int nrApelari=0;
    public static void main(String[] args) {
        System.out.println("5>4: "+(5>4));
        System.out.println("5<4: "+(5<4));
        System.out.println("5>4: "+(5>4 & f()));
        System.out.println("5>4: "+(5>4 && f()));
        System.out.println("5<4 :"+ (5<4 & f()));
        System.out.println("5<4 :"+ (5<4 && f()));
    }
    static boolean f(){
        System.out.println("Se executa f a"+(++nrApelari)+"-a oara");
        return true;
    }
}
```

Facilitatea de scurtcircuitare poate fi folosita in doua moduri:

- ca forma de optimizare. Ea nu este de obicei utila atunci cand operanzii sunt simple variabile obisnuite sau literale (caci esfertul evaluarii operandului 2 este minim), ci atunci cand cel de-al doilea operand este o expresie mai complexa sau un apel de functie/metoda, caz in care neevaluarea sa poate scuti multe operatii din partea masinii virtuale
- ca forma de evaluare treptata a unor expresii care se interconditioneaza. Sa consideram exemplul urmator:

```
String[] nume;      // un array de obiecte String
if(nume != null && nume.length >0 && nume[0] !=null && nume[0].length()>0) {
    System.out.println(nume[0]);
```

Instinctiunea decizionala if primeste ca argument o expresie formata din patru conditii elementare. Variabila *nume* este o referinta catre un obiect de tip tablou (vezi capitolul urmator). Aceasta poate insa sa nu existe (caz in care *nume* are valoarea null); in acest caz, incercarea de a verifica a doua conditie (*nume.length>0*) s-ar solda cu o eroare. Grati la scurtcircuitarii, *nume.length* va fi accesat numai daca *nume* nu are valoarea null. In continuare, chiar daca tabloul exista

(conditia 1 indeplinita), conditia 3 ar produce o eroare daca tabloul ar fi gol; din fericire, conditia 3 se va evalua numai daca primele doua sunt true (asadar tabloul exista si contine cel putin un element). In sfarsit, ultima conditie depinde si ea de indeplinirea tuturor celor precedente (tabloul trebuie sa existe, sa contina cel putin un element si acel element sa fie diferit de null), si scurtcircuitarea ne asigura ca accesarea lungimii primului element nu va rezulta intr-o eroare.

1.4.8. Operatorul ternar conditional

Acest operator produce una din doua valori posibile, in functie de valoarea unei expresii de tip boolean. Toate cele trei elemente sunt specificate ca operanzi de catre programator. Sintaxa sa este:

```
conditie?valoare_daca_conditia_e_adevarata:valoare_daca_conditia_e_falsa
```

Conditia este o expresie care trebuie sa se evalueze ca boolean. Cele doua valori pot fi literals sau rezultatele unor expresii, cu urmatoarele conditii:

- expresiile trebuie sa produca o valoare. Spre exemplu, nu putem folosi pe post de operand 2 sau 3 apelul System.out.println()
- atunci cand valoarea produsa de operator este atribuita unei variabile, trebuie ca valorile operanzilor 2 si 3 sa fie assignment-compatible cu tipul de date al variabilei destinatie

Operatorul ternar se foloseste pentru a implementa o logica decizionala simpla, ce poate fi intercalata in cadrul unei expresii (spre deosebire de cazul instructiunilor decizionale explicite, *if* sau *switch*). Exemple de utilizare:

```
public class c4 {
    public static void main(String[] args) {
        int x=2,y=3;
        System.out.println("x: "+x+", y : "+y);
        System.out.println("x>y? "+(x>y?"adevarat":"fals"));
        String paritateX = ((x%2==0)? "par": "impar");
        System.out.println("Numarul x este "+paritateX);
    }
}
```

1.5. BIBLIOGRAFIE

- Lista de termeni/acronime uzuale Java: <http://www.oracle.com/technetwork/topics/newtojava/unravelingjava-142250.html>
- Download JDK pentru Java SE: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Tipuri de date primitive: <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>
- Operatori: <http://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html>

1.6. Anexa: Ghidul studentului InfoAcademy – primii pasi

Asociația pentru educație integrată în domeniul IT

InfoAcademy-Cisco Networking Academy

e-mail: ionut@infoacademy.net

web: www.infoacademy.net

Accesul la sistemul educational InfoAcademy

Interacțiunea cu sistemul educational InfoAcademy se face “on-line” și “in-person”.

- Accesul la serviciile on-line ale InfoAcademy se face prin <http://www.infoacademy.net>
- Accesul la serviciile “in-person” oferite de InfoAcademy (predare, laboratoare, consultanță) se face prin instructorii de curs.

Crearea contului

- Dacă nu aveti cont InfoAcademy, după prima sedință veti primi pe e-mail informațiile de login, care va vor permite accesul pe www.infoacademy.net.
- Se accesează <http://www.infoacademy.net>
- La mijlocul paginii, deasupra tabelului cu clase se află cele două campuri pentru username și parola, unde folositi informațiile de login primite
- Contul va permite să intrati în zona protejată a site-ului InfoAcademy
 - După login vi se afisează tabelul de extraservicii
 - Selectați link-ul “Student Home”
 - Veti vedea clasa în care sunteți înscris(a), cu link-uri către documentație aditională, examene etc

Modificarea datelor din contul InfoAcademy

- Poate fi făcută după login pe [infoacademy.net](http://www.infoacademy.net) accesând linkul “Editare profil” prezent în dreapta sus, alături de cel de logout
- Permite modificarea e-mailului, telefonului și datei de nastere
- **ATENTIE !!** Completati corect numele, prenumele și initiala tatălui (acestea sunt tiparite pe certificatul obținut la promovarea cursului)
- **ATENTIE !!** Introduceti în cont adresa de e-mail cea mai utilizată și actualizați-o ori de câte ori o modificați. Pe această adresa va veni comunicările oficiale de la InfoAcademy

Documentația

- Documentația cursului este formată din:
 1. documentație în format electronic, în limba română
 2. documentație suplimentară în format electronic, în limba engleză
- Documentația în format electronic
 1. este disponibilă din contul de student, prin link-ul Documentație din cadrul clasei, secțiunea Materiale de Curs
 2. este împărțită în 13 capitole
 3. studiul ei este OBLIGATORIU și presupune asimilarea a cel puțin 1 capitol/săptămână
- Documentația aditională (în general în limba engleză)
 1. este disponibilă din contul de student, prin link-ul Documentație aditională
 2. studiul ei este optional dar recomandat
- Documentație suplimentară, utilă pentru aprofundarea/imbogătirea cunoștințelor, este accesibilă on-line 24 de ore din 24 folosind link-ul “Documentație aditională” din pagina clasei în care sunteți înscris(a)

Examenele

- sunt de 4 tipuri: partiale, mid-term, final, feedback
- **examenele partiale**
 - sunt obligatorii si se dau pe <http://www.infoacademy.net>
 - sunt in limba engleza
 - sunt de tip grila cu raspuns simplu sau multiplu
 - dureaza maxim 1h
 - sunt accesibile (dupa ce intrati pe clasa activa) prin linkul "Take Assessment"
 - se pot da de maxim 3 ori
 - sunt strict din capitolul corespunzator al documentatiei online (ex: examenul 1 corespunde capitolului 1)
 - sunt active NUMAI in zilele si la orele anuntate de instructor:
 - duminica 16:00 – marti 16:00
 - marti 16:00 - joi 16:00
 - joi 16:00 – sambata 16:00
 - se considera promovate daca punctajul este de cel putin 75%
 - fiecare examen are o pondere in nota finala, pondere afisata in Gradebook
- **examenul "Mid-term"**
 - se sustine NUMAI de la Academie in prezenta doamnei/d-lui Marinescu sau a instructorului de curs, imediat dupa sedinta a 6-a
 - sustinerea are loc vineri dimineata intre orele 8 si 12 si presupune programare online prealabila pe www.infoacademy.net
 - promovarea acestui examen este conditie de continuare a cursului
 - se da din materia capitolelor 1-7
 - se poate sustine de maxim 2 ori
 - este ultima ocazie de a achita cea de-a doua rata, daca este cazul
- **examenul final**
 - se da NUMAI de la Academie in prezenta doamnei Mihaela Marinescu sau a unui instructor delegat, la cel mult 3 saptamani de la data ultimei sedinte de curs
 - se poate sustine de maxim 2 ori
 - se sustine numai vinerea dimineata intre 8 si 12
 - pot participa la examen numai cei care au promovat celelalte examene si s-au inscris la examen prin formularul online de pe www.infoacademy.net
- **examenul Course Feedback**
 - va cere impresii despre curs, materiale, laboratoare, instructor , etc
 - nu se puncteaza
 - este obligatoriu pentru inchiderea situatiei
- **ATENTIE !!** Daca in timp ce dati un examen va cade Internetul puteti reintro pe examen NUMAI daca o faceti in cadrul aceleiasi sesiuni de activare. Ex: daca dati examenul de la capitolul 1 duminica de la 17 si la 17:15 va cade conexiunea internet , puteti reintro pe examen pana marti la ora 15:15 pentru a continua cele 45 minute ramase. In caz contrar examenul se blocheaza si va trebui o aprobatie speciala (veti fi chemat la academie sa dati examenul)
ATENTIE !! Orice ramanere in urma cu mai mult de 2 examene atrage dupa sine exmatriculararea fara preaviz si pierderea oricaror drepturi de a continua acel modul.

Catalogul electronic

- Catalogul electronic "Gradebook"
 - Se afla pe site-ul [infoacademy.net](http://www.infoacademy.net)

- Inregistreaza rezultatele examenelor pe care le dati on-line
- Dupa promovarea tuturor examenelor cu cel putin 75% , instructorul va inchide situatia – care este pusa in evidenta printr-o litera din Gradebook :
 - Student inscris – “E”
 - Student promovat cu situatia inchisa - “P”
 - Student care din cauze de Forta majora (demonstrata legal) nu a terminat modulul - “I”
 - Student cu situatia inchisa si NEPROMOVAT - “F”

Proiectul final

- Presupune elaborarea unei aplicatii folosind cunostintele dobandite de-a lungul cursului
- Este conditie de incheiere a cursului
- Enuntul sau este trimis de catre instructor prin e-mail cand se apropie finalul de curs
- Rezolvarea se trimit tot prin email; instructorul evalueaza proiectul si decide daca acesta este admis sau respins

Diploma

- Trebuie solicitata online, folosind formularul corespunzator din sectiunea de servicii a site-ului www.infoacademy.net
- Pentru a depune solicitarea, trebuie ca studentul sa fi promovat toate examenele cu minim 75% (cele de capitol, mid-term, final), sa fi dat Course Feedback si sa-i fi fost acceptat proiectul final
- In urma depunerii cererii, academia emite diploma semnata si stampilata si notifica studentul prin e-mail cand poate veni sa o ridice de la sediul academiei
- In cazul in care cererea este incompleta sau studentul nu indeplineste conditiile pentru eliberarea diplomei, instructorul marcheaza cererea ca fiind invalida, anuntand studentul sa ia masurile necesare. Aceasta poate sterge cererea si depune una noua, corecta, folosind acelasi formular on-line

Serviciile on-line oferite de InfoAcademy

- Se acceseaza prin pagina privata a www.infoacademy.net la care aveti acces utilizand contul primit
- Permit :
 - Consultanta on-line (se pot cere explicatii referitoare la problemele teoretice, practice si examene)
 - Feedback (opinii, recomandari, nemultumiri exprimate pe tot parcursul cursului)
 - Cerere de activare examene (pentru situatii speciale care nu permit studentului sustinerea examenelor in zilele default)
 - Cerere de diploma (cel ce a absolvit un modul cere diploma, care va emisa de catre InfoAcademy, semnata de catre instructorul de curs, stampilata si inmanata studentului)
 - Inscrisire la examenul final (examenul final se da la academie pe baza programarii facute)
 - Inscrisire in programul “Sustine Performanta” (pentru cei care, in urma notelor obtinute la cursurile InfoAcademy, sunt eligibili pentru acest program)

Serviciile in-person oferite de InfoAcademy

- Sunt accesibile prin instructorii de curs.
- Constat in:
 - Predarea aprofundata a programei cursului
 - Laboratoare explicative, ilustrative si de fixare de cunostinte
 - Consultanta pentru nelamuririle referitoare la notiunile incluse in examene.