Laborator 2 – Bazele Programarii Orientate pe Obiect

1. Cum ne organizam un proiect Java.

Înainte de a trece la implementare trebuie sa ne gandim cum ne structuram din punct de vedere logic programul pe unitati (grupuri). Elementele care fac parte din aceeasi grup trebuie să fie **conectate în mod logic**, pentru o ușoară implementare și înțelegere ulterioară a codului. În cazul limbajului Java, aceste grupuri care contin entitati (ex. clase, interfete) conectate din punct de vedere logic intre ele poarta numele de pachete. In catrul unui proiect pachetele sunt organizate intr-o structura ierahica, putand forma o ierarhie pe mai multe nivele (ex. pachete in pachete).

Clasele pot fi create fiecare separat in cate un fisier, sau puteam avea definite mai multe clase in cadrul aceluiasi fisier respectand urmatoarea reguli:

1. dacă dorim ca această clasă să fie vizibilă din întreg proiectul, îi vom pune specificatorul **public** (vom vorbi despre specificatori de acces mai în detaliu în cursurile urmatoare); acest lucru implică însă 2 restricții:
   - fișierul și clasa publică trebuie să aibă același nume
   - nu poate exista o altă clasă/interfață publică în același fișier (vom vedea în laboratoarele următoare ce sunt interfețele)
2. pot exista mai multe clase în același fișier sursă, cu condiția ca **maxim una** să fie publică

2. Crearea documentației unui proiect Java.

Pentru a genera automat documentația unui proiect Java accesați meniul Project->Generate Javadoc… Selectați toate fișierele cu cod sursă Java pentru care doriți să se genereze documentație. Acestea trebuie să conțină în prealabil tag-uri Javadoc /** … */ al căror conținut va apărea în documentația generată. În urma generării documentației rezultă în principal o ierarhie de fișiere HTML. Pentru a vizualiza documentația generată deschideți fișierul principal index.html.

3. Consideratii teoretice ajutatoare pentru implementarea temei.

a. Numere aleatorii. In java avem la dispozitie doua variante pentru a genera numere aleatorii:

Folosind clasa **Random**. Pentru a utiliza această clasă pentru a genera numere aleatorii, trebuie mai întâi să creăm o instanță a clase și apoi să invocăm metode precum nextInt (), nextDouble (), nextLong () etc. folosind acea instanță. Prin intermediul metodelor clasei putem genera numere aleatorii de tipuri întregi, flotante, duble, lungi, boolean, etc. Ca argument al metodelor se transmite limita superioara corespunzatoare numerelor care urmeaza a fi generate. De exemplu, nextInt (6) va genera numere cuprinse între 0 și 5, inclusiv 0 si 5.

Exemplu de utilizare a clasei Random:

```
import java.util.Random;
…….
Random rand = new Random();

// Generate random integers in range 0 to 999
int rand_int1 = rand.nextInt(1000);
int rand_int2 = rand.nextInt(1000);

// Generate Random doubles
```

```
double rand_dub1 = rand.nextDouble();
double rand_dub2 = rand.nextDouble();
```

Folosind **Math.random()**. Clasa **Math** conține diverse metode pentru efectuarea de diferite operații numerice, cum ar fi, calculul ridicarii la putere, logarithm, etc. Una dintre aceste metode este **random**(). Această metodă returnează o valoare de tip **double**, mai mare sau egal cu 0,0 și mai mic decât 1,0. Valorile returnate sunt alese pseudoaleatoriu.

Exemplu de utilizare a metodei random() din clasa Math:

```
// Generating random doubles
System.out.println("Random doubles: " + Math.random());
System.out.println("Random doubles: " + Math.random());
```

b. **Citirea datelor din consola.** Pentru citirea datelor din consola in java exista mai multe variante:

Folosind clasa **Scanner.** Aceasta este probabil cea mai preferată metodă de preluare a datelor. Scopul principal al clasei **Scanner** este de a analiza tipurile și șirurile primitive folosind expresii regulate, cu toate acestea, se poate folosi și pentru a citi intrările date de utilizator în linia de comandă.

Exemplu de utilizare a clasei Scanner:

```
import java.util.Scanner;
// Using Scanner for Getting Input from User
Scanner in = new Scanner(System.in);

String s = in.nextLine();
System.out.println("You entered string " + s);

int a = in.nextInt();
System.out.println("You entered integer " + a);

float b = in.nextFloat();
System.out.println("You entered float " + b);
```

Folosind clasa **Console.** Clasa poate fi folosita pentru citirea informațiilor date de utilizator din linia de comandă:

Exemplu de utilizare a clasei Console:

```
// Using Console to input data from user
  String name = System.console().readLine();
System.out.println("You entered string " + name);
```

c. **Metode recursive**. O metodă care se numește se apeleaza pe ea insasi se numeste metodă recursivă. Acest proces este cunoscut sub numele de recursivitate. Folosind un algoritmul recursiv, anumite probleme pot fi rezolvate mult mai ușor.

```
    public static void main(String[] args)
     {
          ……
          method1();
          ……
          }
 static void method1 ()
  {
    ……
      method1();
    ……..
 }
```

In exemplul de mai sus, metoda method1 () a fost aplelata din interiorul metodei main(). In acest caz vorbim de un apel normal al metodei. In acelasi timp in interiorul metode method1(), este aplelata acceasi metoda method1(). Acesta este un exemplu de apel recursiv. Pentru a iesi din recursiv, trebuie puse niste conditii suplimentare în cadrul metodei. În caz contrar, metoda va rula la infinit. O modalitate este folosirea structurii **if** ... **else** (sau o abordare similară) pentru a termina apelul recursiv din interiorul metodei.

Exemplu:

```
public class RecursionExample2 {
static int count=0;
static void p(){
 count++;
          if(count<=5){
            System.out.println("hello "+count);
             p();
  }
}
public static void main(String[] args) {
   p();
 }
}
```

Probleme de rezolvat:

1. Scrieti o clasa TestAritmetic care genereaza doua numere aleatoare intregi si afiseaza la consola: cele doua numere, suma/ diferenta celor doua numere, inmultirea celor doua numere, impartirea intrega, restul impartirii celor doua numere.

2. Scrieti a aplicatie care calculeaza radacina patrata, folosind metoda Newton

Algoritmul de calcul pentru sqrt(Number) folosind metoda lui Newton este:

1. Se seteaza x=1
2. Se seteaza prag = 0.01
    a. Se calculeaza x1 = 1/2 (x+ Number/x)
    b. Se verifica daca |x1 – x| < prag
        i. Daca da, se iese
    c. Se seteaza x = x1

d. Se repeta pasul 2.a

3. Se returneaza valoarea lui x

3. Scrieti o aplicatie java care defineste doua metode: (1) metoda java care sa returneze minimul a trei numere de acelasi tip primitiv (folosind operatorul conditional? :), (2) o metoda java care returneaza minimul dintre patru numere si care foloseste in implementare metoda anterior definita. Utilizatorul poate specifica de la tastatura daca doreste sa calculeze minimul pentru trei sau patru numere, iar in functie de optiunea aleasa va introduce numerele corespunzatoare pentru care vrea sa se calculeze minimum. Minimum se va afisa in consola.

4. Sa se scrie o aplicatie care caculeaza suma de bani detinuta de o persoana dupa un anumit numar de an i(se va face capitalizarea contului, adica se va tine cont de "dobanda la dobanda"). Se presupune ca pentru suma depusa (depozit la temen) dobanda oferita de catre banca este de 25% pe an. Pentru aceasta aplicatie datele (ex. suma depusa, numarul de ani) sunt introduse de catre utilizator de la tastatura.Aplicatia va afisa in consola suma detinuta de client dupa un anumit numar de ani.

5. Să se scrie un program care citeşte de la tastatura valoarea reală a lui x şi calculează valoarea funcţiei de mai jos.

$$f(x)=\begin{cases} x^2+4x+4 & daca \quad x<0 \\ 0 & daca \quad x=0 \\ x^2+5x & daca \quad x>0 \end{cases}$$

6. Sa se scrie un program Java care calculeaza n! (n!=!*2*...*n), unde 0<n<13 este un numar natural. Pentru implementarea se va defini o metoda recursiva. Aplicatia citeste de la tastatura numarul pentru care se calculeaza factorialul.