ISw - Seminar 1.

Introducere in limbajele Java (de programare OO) si UML (de modelare OO)

1.1. Descrierea seminarului

In acest seminar vor fi acoperite total sau partial urmatoarele probleme:
- **structura unui program** Java, **stiluri de scriere a comentariilor**
- **atribute, variabile locale, metode** (declaratie, invocare, pasarea parametrilor, returnarea valorilor)
- **diagrame UML de clase si secventa** (a mesajelor)

1.2. Probleme rezolvate si probleme propuse spre rezolvare

Problema 1.1.

Se da urmatorul program simplu Java.

```
1    public class Salut1 {
2      public static void main(String[] args) {
3        System.out.println(“Buna ziua!”);
4      }
5    }
```

a) Descrieti programul, linie cu linie.

Rezolvare:

O **posibila descriere a programului, linie cu line**, este urmatoarea:

Linia 1 (prima linie) din program declara o **clasa Java** (cf. `class`), numita `Salut1`, **al carei cod poate fi accesat de orice cod exterior ei** (cf. `public`). Dupa declaratia clasei, urmeaza corpul ei, aflat intre elementele operatorului de declarare a blocurilor (acolade: “`{`“ si “`}`”).

Linia 2 din program **declara o metoda** `main()`, care este **metoda cu caracter global**, de clasa (cf. `static`) si **nu returneaza nici o valoare** (cf. `void`). Metoda `main()` este **numita *punct de intrare in program***, si reprezinta **metoda care va fi executata prima, atunci cand va fi lansata interpretarea clasei** `Salut1`.

Metoda `main()` **primeste ca parametru**, in interiorul operatorului listei de parametric ai metodelor (paranteze rotunde: “`(`“ si “`)`”), **un tablou de obiecte de tip** `String`. Operatorul de indexare (paranteze drepte: “`[`“ si “`]`“), este folosit pentru a declara tablouri. **Prin intermediul acestui tablou, interpretorul Java paseaza argumentele adaugate de utilizator dupa numele interpretorului (*`java`*) si al programului (clasei, in cazul nostru, *`Salut1`*).**

Programul poate utiliza sau nu aceste argumente, pe care **le acceseaza prin intermediul referintei** `args` (referinta la tablou de obiecte de tip `String`).

`String` este numele unei clase din biblioteca standard Java (*`java`*), din **pachetul de clase care sunt implicit importate** (*`java.lang`*). **Numele sau complet (calificat de numele pachetului) este**

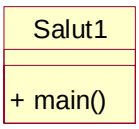
***java.lang.*`String`**, iar **rolul sau este de a reprezenta (incapsula) siruri de caractere Java** (in Java caracterele sunt reprezentate in **format UNICODE**, in care **fiecare caracter necesita 2 octeti pentru codificare**).

Linia 3 din program **reprezinta corpul metodei `main()`**. Ea declara o **instructiune Java de tip invocare de metoda** (apel de functie). Este **invocata metoda `println()` pentru a se trimite pe ecran (in consola standard de iesire) un sir de caractere**. Metoda `println()` **apartine obiectului `out`** (de tip *`java.io.`*`PrintWriter`), care este **atribut cu caracter global**, de clasa, **al clasei** [`java.lang.`]**`System`**. Obiectul `out` corespunde consolei standard de iesire. Operatorul de calificare a numelor (punct: "`.`"), este folosit pentru a se specifica numele calificat al metodei `println()`. Sirul de caractere care ii este pasat ca parametru (**`Buna ziua!`**) este plasat in operatorul de declarare a sirurilor de caractere (ghilimele: "`"`" si "`"`"). **Metoda `println()` nu returneaza nici o valoare.** Instructiunea se incheie cu operatorul de sfarsit de instructiune (punct si virgula: "`;`").

Linia 4 din program precizeaza incheierea declaratiei metodei `main()`.
Linia 5 din program precizeaza incheierea declaratiei clasei **`Salut1`** (liniile 3, 4 si 5 din formeaza corpul clasei `Salut1`) si prin urmare incheierea programului.

b) Creati diagramele UML de clase si de secventa (a mesajelor schimbate) corespunzatoare codului dat.

Rezolvare:



sau, **detaliind semnatura metodei**

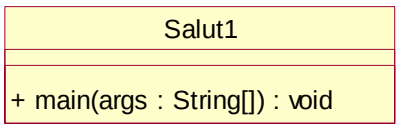
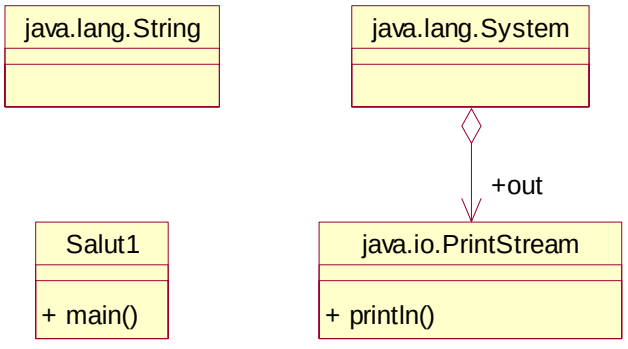
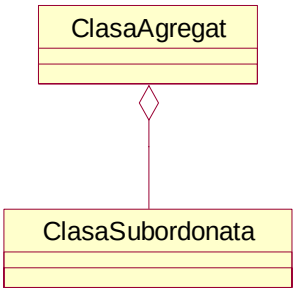


Diagrama contine **simbolul unei singure clase**, caseta de sus continand **numele** ei – in cazul nostru **`Salut1`**, caseta din mijloc **eventualele atribute (variabile membru)** – in cazul nostru nu exista, iar caseta de jos **metodele (functiile membru)** – in cazul nostru metoda **`main()`**, publica (conform simbolului "**`+`**").

O diagrama de clase mai detaliata, care sa **cuprinda si clasele sau obiectele din bibliotecile de clasa Java**, este urmatoarea:



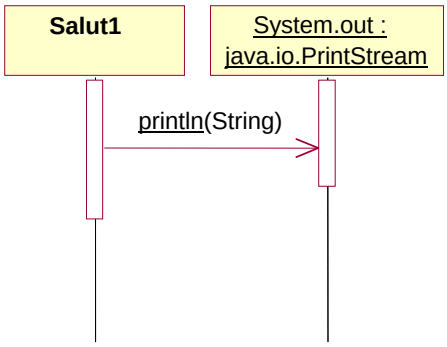
notatie utilizata:



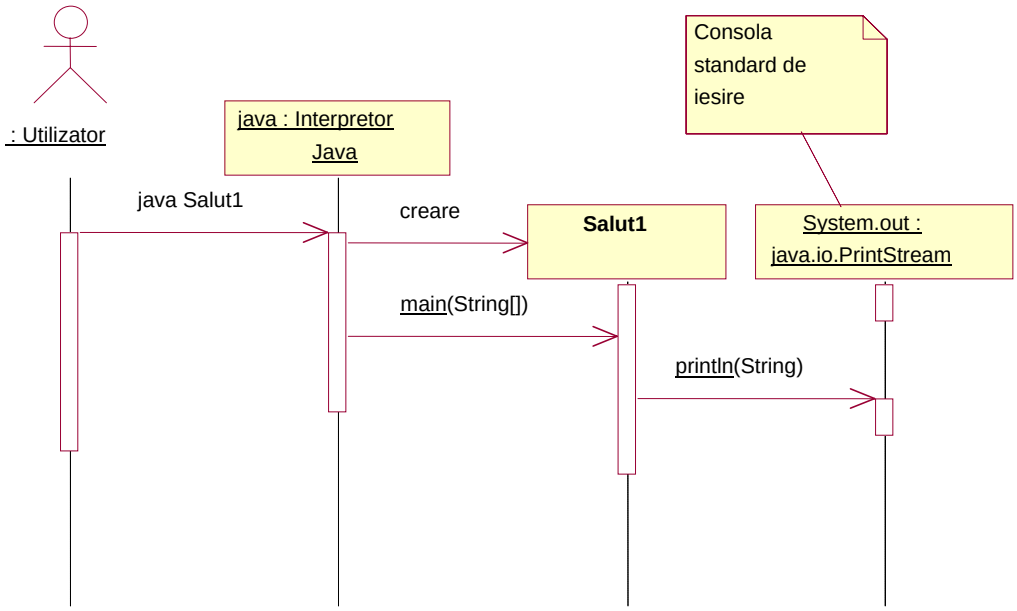
Simbolul sub forma de romb semnifica **relatia de agregare a claselor**, adica o relatie de **subordonare** a unui obiect al **clasei subordonate `PrintStream`** (in cazul de fata obiectul **`out`**) fata de un obiect al **clasei agregat `System`**.

Sageata indica **navigabilitate unidirectionala**, adica faptul ca doar clasa **`System`** are acces la obiecte din clasa **`PrintStream`** (prin constructie clasa **`System`** poate accesa un obiect al clasei **`PrintStream`**, si anume **`out`**) pe cand clasa **`PrintStream`** nu are acces la nici un obiect al clasei **`System`** (tot prin constructie).

Diagrama de secventa care **ilustreaza schimbul de mesaje intern programului** este urmatoarea.



O posibila diagrama de secventa detaliata, in care este **ilustrat si procesul lansarii in executie**, este urmatoarea.



Problema 1.2.

Sa se modifice programul Java de la problema **Problema 1.1.** astfel incat:

a) mesajul `Buna ziua!` sa nu fie pasat direct metodei **`println()`**, ci prin intermediul unui **atribut al clasei `Salut2a`**, de tip **`String`**, avand numele **`atributString`**;

b) mesajul `Buna ziua!` sa nu fie pasat direct metodei **`println()`**, ci prin intermediul unei **variabile locale metodei `main()`** a clasei **`Salut2b`**, de tip **`String`**, avand numele **`variabilaLocalaString`**;

c) mesajul pasat metodei **`println()`** sa fie **primul parametru primit** de metoda **`main()`** a clasei **`Salut2c`** **din linia de comanda** (in momentul executiei interpretorului **`java`** asupra clasei **`Salut2c`**).

d) Creati diagramele UML de clase corespunzatoare codului rezultat.

Rezolvari:

a) Posibile modificari ale programului de la problema **Problema 1.1.** astfel incat mesajul **`Buna ziua!`** **sa fie pasat metodei `println()` prin intermediul unui atribut al clasei `Salut2a`**, de tip `String`, avand numele `atributString`, sunt urmatoarele:

I. Forma cea mai complexa (codul modificat este prezentat in format intensificat - *bold***):**

```
 1    public class Salut2a {
 2
 3      public static String atributString;          // declarare atribut
 4                                                   // cu caracter global
 5      public static void main(String[] args) {
 6
 7        atributString = new String("Buna ziua!"); // alocare si initializare
 8
 9        System.out.println(atributString);        // utilizare atribut
10      }
11    }
```

II. Linia 7 din varianta I este **echivalenta cu:**

```
        atributString = "Buna ziua!";   // alocare si initializare rapida,
                                        // posibila doar pentru clasa String
```

III. Linia 3 din varianta I poate fi rescrisa astfel (caz in care **linia 7 din varianta I dispare**)**:**

```
        // declarare, alocare si initializare intr-o singura linie de cod
        public static String atributString = new String("Buna ziua!");
```

IV. Linia 3 din varianta I poate fi rescrisa si (caz in care **linia 7 din varianta I dispare**)**:**

```
        // declarare, alocare si initializare intr-o singura linie de cod
        public static String atributString = "Buna ziua!";
```

Observatie:

Variantele anterioare utilizeaza un **atribut cu caracter `static`** (global, locatie unica la nivel de clasa, partajata de toate obiectele clasei). Daca insa atributul ar fi declarat fara cuvantul cheie **`static`**, in linia 3, atunci ar fi generate erori la compilare in liniile 7 si 9 :

```
directorcurent>javac Salut2a.java
Salut2a.java:7: non-static variable atributString cannot be referenced
from a static context
      atributString = new String("Buna ziua!");
      ^
Salut2a.java:9: non-static variable atributString cannot be referenced
from a static context
      System.out.println(atributString);
                         ^
2 errors
```

In acest caz, **problema** provine din faptul ca **atributul declarat non-static este creat la nivel de individual, fiecare obiect avand propria locatie** cu numele **`atributString`**. **Accesul la locatia `atributString` a unui obiect particular `salut`** se face utilizand **sintaxa `salut.atributString`.**

O posibila solutie este crearea unui obiect **`salut`** al clasei **`Salut2a`**, urmata de utilizarea atributului **`salut.atributString`**:

```
3      public String atributString;              // declarare atribut
4                                                // cu caracter individual
5      public static void main(String[] args) {
6        Salut2a salut = new Salut2a();
7        salut.atributString = new String("Buna ziua!");
8
9        System.out.println(salut.atributString);
```

In noua linie 6 se observa aparitia unei "metode" speciale, **`Salut2a()`**, avand acelasi nume cu al clasei, **care poarta denumirea de constructor** (si, in general, rolul de a initializa atributele).

Compilatorul Java creeaza un constructor fara parametri si fara implementare atunci cand programatorul nu declara in mod explicit unul (**acesta este si cazul nostru**).

b) O posibila modificare a programului de la problema **Problema 1.1.** astfel incat mesajul **`Buna ziua!` sa fie pasat metodei `println()` prin intermediul unei variabile locale metodei `main()`**, de tip `String`, avand numele **`variabilaLocalaString`**, este urmatoarea:

```
1    public class Salut2b {
2
3      public static void main(String[] args) {
4
5        String variabilaLocalaString = "Buna ziua!"; // declarare, alocare si
6                                                     // initializare variabila
7        System.out.println(variabilaLocalaString);
8      }
9    }
```

c) O posibila modificare a programului de la problema **Problema 1.1.** astfel incat mesajul **pasat metodei `println()` sa fie primul parametru primit de metoda `main()` din linia de comanda**, este urmatoarea:

```
1    public class Salut2c {
2
3      public static void main(String[] args) {
4
5        System.out.println(args[0]);
6      }
7    }
```

Observatie:

In cazul in care utilizatorul nu adauga un argument in momentul lansarii in executie a clasei, tabloul **`args` va fi** un **tablou vid (fara elemente)**, iar elementul **`args[0]` nu va exista**. Efectul va fi **generarea unei exceptii Java din clasa** `java.lang.`**`ArrayIndexOutOfBoundsException`**.

Mesajul obtinut in momentul lansarii in executie va fi de forma :

```
directorcurent>java  Salut2c
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
     at Salut2c.main(Salut2c.java:5)
```

O posibila solutie este urmatoarea:

```
1    public class Salut2c {
2      public static void main(String[] args) {
3        if (args.length != 1) {
4          System.out.println("Utilizare program: java Salut2c Salut!");
5        }
6        else {
7          System.out.println(args[0]);
8        }
9      }
10   }
```

Mesajul obtinut in momentul lansarii in executie va fi de data asta de forma :

```
directorcurent>java  Salut2c
Utilizare program: java Salut2c Salut!
```

O alternativa este **tratarea exceptiei `ArrayIndexOutOfBoundsException`** cu ajutorul unui bloc:

```
try {
   // secventa care poate genera exceptia
}
catch (ArrayIndexOutOfBoundsException e) {
   // secventa care trateaza exceptia
}
```

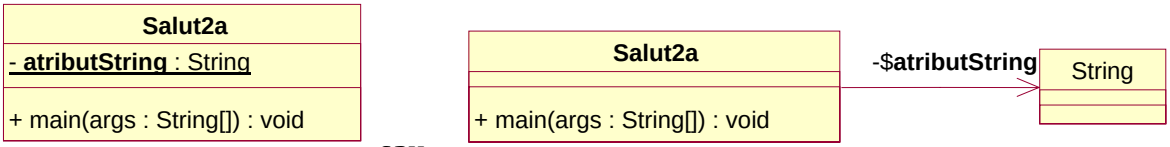
In acest caz codul ar putea arata astfel:

```
1    public class Salut2c {
2      public static void main(String[] args) {
3        try {
4          System.out.println(args[0]); // secventa care poate genera exceptia
5        }
6        catch (ArrayIndexOutOfBoundsException e) {
7          // secventa care trateaza exceptia
8          System.out.println("Utilizare program: java Salut2c Salut!");
9        }
10     }
11   }
```

Mesajul obtinut in momentul lansarii in executie este si de data aceasta de forma :

```
directorcurent>java  Salut2c
Utilizare program: java Salut2c Salut!
```

d) Diagrama de clase pentru clasa **`Salut2a`** poate avea doua forme:



Se observa ca **atributul** **`atributString`**, de tip `String`, **poate fi interpretat si ca asociere (cu navigabilitate unidirectionala) intre clasele** **`Salut2a`** **si** **`String`**.

Diagramele de clase pentru celelalte clase sunt urmatoarele:

Problema 1.3.

a) Sa se adauge programului Java de la problema **Problema 1.2.a** **o** **metoda** **`afisareAtributString()`** care sa trimita pe ecran (in consola standard de iesire) atributul **`atributString`**. **Sa se modifice** metoda **`main()`**astfel incat sa foloseasca **metoda** **`afisareAtributString()`** pentru a afisa atributul **`atributString`**.

b) Sa se adauge programului Java de la problema **Problema 1.2.a** **o** **metoda** **`afisareString()`** care sa afiseze pe ecran un sir de caractere de tip **`String`**. **Sa se modifice** metoda **`main()`**astfel incat sa foloseasca **metoda** **`afisareString()`** pentru a afisa atributul **`atributString`**.

c) Creati diagramele UML de secventa corespunzatoare codului rezultat.

Rezolvari

a) O posibila modificare a programului de la problema **Problema 1.2.a** astfel incat sa i se **adauge o metoda `afisareAtributString()`** care sa afiseze pe ecran atributul **`atributString`**, iar metoda **`main()`** sa foloseasca **metoda `afisareAtributString()`** pentru a afisa atributul **`atributString`**, este urmatoarea:

```
1   public class Salut3a {
2     public static String atributString;      // atribut cu caracter global
3
4     public static void afisareAtributString() { // metoda cu caracter
5       System.out.println(atributString);        // static (global)
6     }
7
8     public static void main(String[] args) {
9       atributString = new String("Buna ziua!");
10      afisareAtributString();
11    }
12  }
```

Observatie:

Varianta anterioara utilizeaza un **atribut cu caracter `static`** (global). Daca insa atributul ar fi declarat fara cuvantul cheie **`static`** atunci ar fi necesara crearea unui obiect **`salut`** al clasei **`Salut3a`**, urmata de utilizarea atributului **`salut.atributString`**. O posibila solutie (care utilizeaza **si o metoda `afisareAtributString()` cu caracter non-static**):

```
1   public class Salut3a {
2     public String atributString;       // atribut cu caracter individual
3
4     public void afisareAtributString() {       // metoda cu caracter
5       System.out.println(atributString);       // non-static (individual)
6     }
7
8     public static void main(String[] args) {
9       Salut3a salut = new Salut3a();
10      salut.atributString = new String("Buna ziua!");
11      salut.afisareAtributString();
12    }
13  }
```

O alternativa (**recomandata** in acest caz) este **declararea si implementarea unui constructor, `Salut3a()`, care sa realizeze initializarea atributului**. **Linia 10 poate sa dispara**. De exemplu:

```
1   public class Salut3a {
2     public String atributString;       // atribut cu caracter individual
3
4     public Salut3a(String s) {{        // constructor - initializeaza
5       atributString = s;               // obiectul (atributele lui)
6     }
7
8     public void afisareAtributString(){       // metoda cu caracter
9       System.out.println(atributString);      // non-static (individual)
10    }
11
12    public static void main(String[] args) {
13      Salut3a salut = new Salut3a("Buna ziua!");
14      salut.afisareAtributString();
15    }
16  }
```

b) O posibila modificare a programului de la problema **Problema 1.2.a** astfel incat sa i se **adauge o metoda `afisareString()`** care sa afiseze pe ecran un sir de caractere de tip **`String`**, iar metoda **`main()`** sa foloseasca **metoda `afisareString()`** pentru a afisa atributul **`atributString`**:

```
1    public class Salut3b {
2      public static String atributString;      // atribut cu caracter global
3
4      public static void afisareString(String s) {  // metoda cu caracter
5        System.out.println(s);                       // static (global)
6      }
7
8      public static void main(String[] args) {
9        atributString = new String("Buna ziua!");
10       afisareString(atributString);
11     }
12   }
```

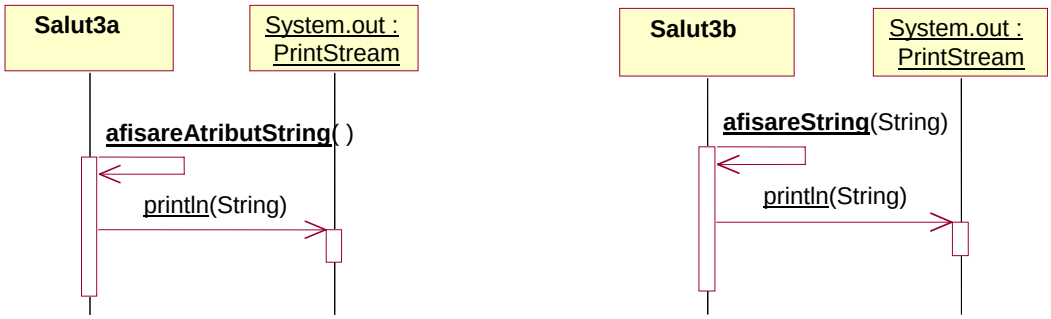
Observatie:

Varianta anterioara utilizeaza un **atribut cu caracter `static`** (global). Daca insa atributul ar fi declarat fara cuvantul cheie **`static`** atunci este necesara crearea unui obiect **`salut`** al clasei **`Salut3b`**, urmata de utilizarea atributului **`salut.atributString`**. O posibila solutie:

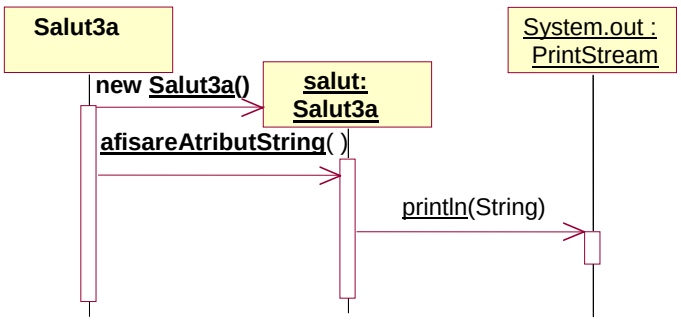
```
1    public class Salut3b {
2      public String atributString;        // atribut cu caracter individual
3
4      public static void afisareString(String s) {  // metoda cu caracter
5        System.out.println(s);                       // static (global)
6      }
7
8      public static void main(String[] args) {
9        Salut3b salut = new Salut3b();
10       salut.atributString = new String("Buna ziua!");
11       afisareString(salut.atributString);
12     }
13   }
```

c) Diagramele de secventa sunt urmatoarele:

- **pentru variantele care utilizeaza metode cu caracter `static`** (global):



- **pentru varianta care utilizeaza metode cu caracter `static`** (global):



Problema 1.4.

Se da urmatorul program Java, numit **`AfisareArgumenteProgram1`**, **care afiseaza argumentele pasate de utilizator din linia de comanda** (in momentul executiei interpretorului `java` asupra clasei `AfisareArgumenteProgram1`)**, argumente pe care le primeste metoda `main()` sub forma de tablou de siruri de caractere**.

```
1    public class AfisareArgumenteProgram1 {
2      public static void main(String[] args) {
3        int i;
4        for ( i=0; i < args.length; i++ ) {
5          System.out.println(args[i]);
6        }
7      }
8    }
```

a) Sa se modifice programul astfel incat sa foloseasca instructiunea `while ... do` in locul instructiunii `for`.

b) Sa se scrie un program `CalculSumaArgumenteIntregi1` pornind de la **programul `AfisareArgumenteProgram1`** care **sa efectueze si sa afiseze, in metoda `main()`, suma valorilor intregi ale argumentelor pasate de utilizator** (urmand ca la utilizatorul sa introduca doar argumente numerice, intregi).

c) Sa se adauge programului `CalculSumaArgumenteIntregi1` tratarea exceptiilor generate atunci cand utilizatorul paseaza argumente care nu sunt intregi.

d) Sa se creeze diagramele UML de clase si de secventa pentru aceste programe.

Problema 1.5.

a) Sa se scrie un program `CalculSumaArgumenteIntregi2`, care sa contina:

- **o metoda** cu caracter **`static`** (globala, la nivel de clasa) numita **`sumaElementeTablouIntregi()`** care:
  - **preia ca parametru un tablou de intregi** (`int[]`),
  - **efectueaza suma elementelor tabloului** si
  - **returneaza valoarea sumei**,
- **o metoda** cu caracter **`static`** numita **`afisareElementeTablouIntregi()`** care
  - **preia ca parametru un tablou de intregi** (`int[]`),
  - **nu returneaza** nimic, si
  - **afiseaza elementele tabloului in formatul**:

```
Elementele tabloului sunt 1 3 5 7 8
Suma elementelor este 24
```

- **o metoda** cu caracter **`static`** numita **`conversieTablouStringIntregi()`** care
  - **preia ca parametru un tablou de siruri de caractere** (`String[]`) si
  - **returneaza un tablou de intregi** (`int[]`) **care contine elementele tabloului primit ca parametru convertite de la sir de caractere** (`String`) **la intregi** (`int`).
- **metoda principala**, `main()`, care
  - **paseaza metodei `conversieTablouStringIntregi()` tabloul de siruri de caractere primit ca parametru** de la utilizator,
  - **paseaza metodei `afisareElementeTablouIntregi()` tabloul de intregi returnat de metoda `conversieTablouStringIntregi()`**, apoi
  - **paseaza metodei `sumaElementeTablouIntregi()` tabloul de intregi returnat de metoda `conversieTablouStringIntregi()`** si
  - **afiseaza valoarea returnata de metoda `sumaElementeTablouIntregi()`**.

b) Sa se creeze diagramele UML de clase si de secventa pentru acest program.