Programare multiparadigmă - **JAVA**

Prof. univ. dr. **Claudiu Vințe**

claudiu.vinte@ie.ase.ro

Tematica

- Introducere în Java. Tipuri de date fundamentale
- Programare orientată obiect în Java
- Tratarea excepțiilor. Java Native Interface
- Java I/O (stream, channel). Serializare
- Java Generics și Java Collections Framework. Adnotări și introspecție
- Programare concurentă
- Programare în rețea. Protocoalele TCP, UDP și HTTP
- Accesul la baze de date prin JDBC
- Utilizare XML și JSON
- Elemente de interfață grafică în JavaFX

Bibliografie

- Ken Arnold, James Gosling, David Holmes, **The Java™ Programming Language**, Fourth Edition, 2005, Addison Wesley Professional, ISBN: 0-321-34980-6
- Daniel Liang, **Introduction to Java Programming**, 11th Edition, Pearson, 2018, ISBN 1-292-22203-4
- Bruce Eckel, **Thinking in Java**, 4th Edition, Prentice Hall, 2006, ISBN 0-13-187248-6
- Joshua Bloch, **Effective Java**, 3rd Edition, Addison-Wesley, 2018, ISBN 0-13-468599-7

- James Gosling, et. al, **The Java Language Specification**, 11th Edition, ORACLE, 2018, https://docs.oracle.com/javase/specs/jls/se11/html/index.html
- Java Platform, Standard Edition & Java Development Kit v11 **API Specification** https://docs.oracle.com/en/java/javase/11/docs/api/index.html
- Raymond Gallardo, et.al., **The Java Tutorial**, ORACLE, 2014 https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html

Evaluare și resurse

- Notare:
  - 70% - examen
  - 30% - seminar

- Resurse:
  - http://online.ase.ro *(parola arhiva cărți: biblio4java)*
  - https://www.w3schools.com/java/ *(limbaj, noțiuni fundamentale)*

Java – caracteristici generale

- Limbaj de uz general
- Limbaj OO imperativ cu elemente funcționale
- Ierarhie aproape unică cu moștenire simplă pentru clase și multiplă pentru interfețe

- Interpretat – bazat pe o mașină virtuală
- Portabil
- Cu gestiune automată a memoriei bazată pe garbage collection

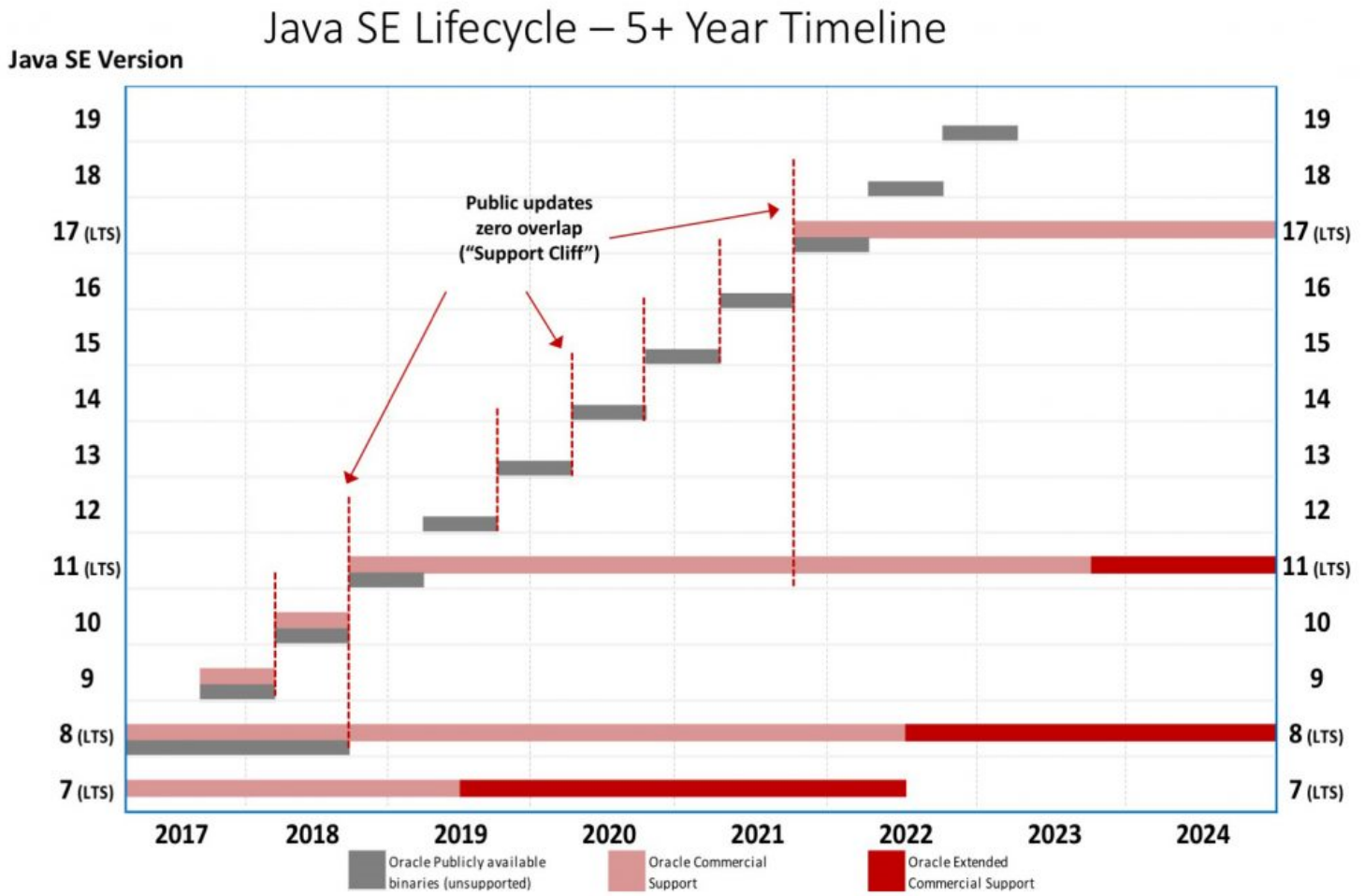
Java – Ediții / Versiuni

Java Platform, Standard Edition (Java SE)

Java Platform, Enterprise Edition (Java EE)

Java Platform, Micro Edition (Java ME)

JavaFX



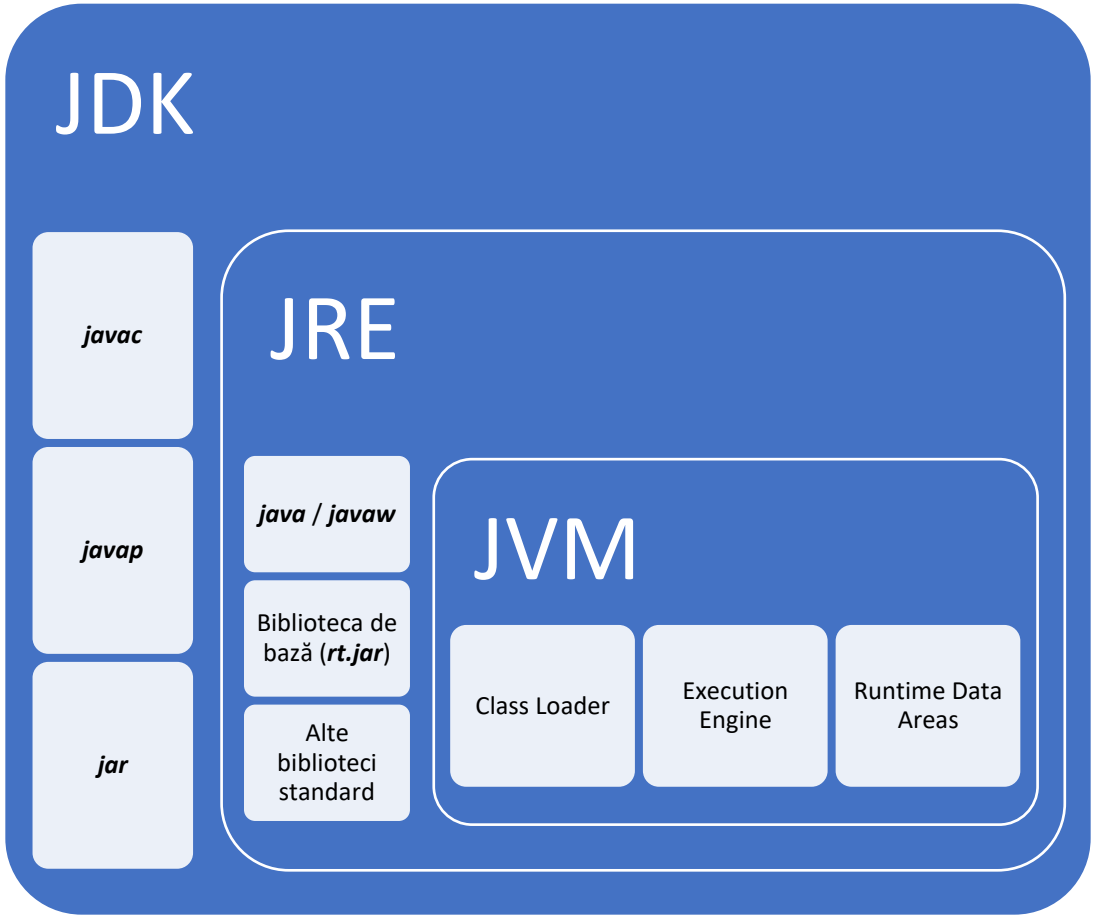
Java – Ediții / Versiuni

Oracle Java SE Support Roadmap*†				
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**	**Sustaining Support**
8 (LTS)**	March 2014	March 2022	December 2030*****	Indefinite
9 - 10 (non-LTS)	September 2017 - March 2018	March 2018 - September 2018	Not Available	Indefinite
11 (LTS)	September 2018	September 2023	January 2032*****	Indefinite
12 - 16 (non-LTS)	March 2019 - March 2021	September 2019 - September 2021	Not Available	Indefinite
17 (LTS)	September 2021	September 2026****	September 2029****	Indefinite
18 - 20 (non-LTS)	March 2022 - March 2023	September 2022 - September 2023	Not Available	Indefinite
21 (LTS)	September 2023	September 2028****	September 2031****	Indefinite
22 (non-LTS)	March 2024	September 2024	Not Available	Indefinite
23 (non-LTS)	September 2024	March 2025	Not Available	Indefinite
24 (non-LTS)***	March 2025	September 2025	Not Available	Indefinite
25 (LTS)***	September 2025	September 2030	September 2033	Indefinite

Source: https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Java – Componentele platformei

- Java Virtual Machine (JVM)
  - Mașina virtuală care rulează codul Java compilat în format *Java bytecode*
- Java Runtime Environment (JRE)
  - Conține instrumentele necesare pentru rularea aplicațiilor Java: mașina virtuală (JVM) și biblioteca standard Java (Java Class Library)
- Java Development Kit (JDK)
  - Conține instrumentele necesare pentru dezvoltarea de aplicații Java precum:
    - javac (*compiler*)
    - jar (*archiver*)
    - javap (*disassembler*)



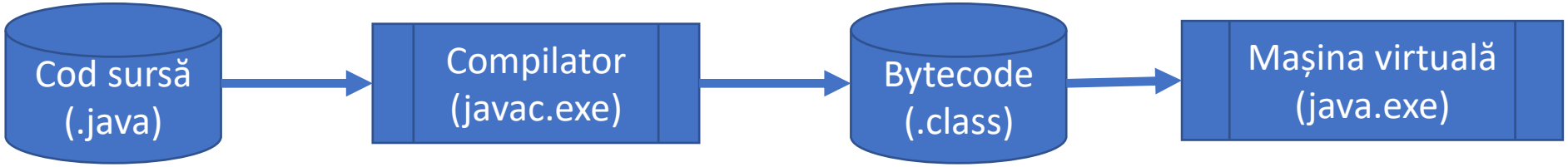
Anatomia unui program Java

- Un program Java este compus dintr-o serie de declarații de tipuri (clase, interfețe, ...)

```
class Program {
    public static void main(String[] args){
        System.out.println("Buna Java!");
    }
}
```

- Punctul de intrare – metoda statică **`main`**:
  - Trebuie să fie declarată publică
  - Trebuie să primească un vector de obiecte **`String`** ca unic parametru
  - Nu trebuie să întoarcă nimic (declarată ca **`void`**)

Procesul de compilare și execuție



1. Compilarea codului folosind ***javac***
   - Primește ca date de intrare
     - Fișierele sursă (fișiere *.java*)
     - Lista căi către bibliotecile utilizate (fișiere *.class* sau *.jar*)
   - Produce fișierul *.class* care conține:
     - Metadate – descrierea completă a tuturor claselor existente în fișier
     - Bytecode – codul executabil pentru fiecare metodă folosind setul de instrucțiuni al JVM
2. Rularea programului folosind mașina virtuală Java - ***java***

Alte instrumente:

jar – construire biblioteci (fișiere *.jar*)

javap – afișare detalii fișiere *.class* (metadate, dezasamblare)

Intrări / ieșiri la consolă

- Accesibile prin intermediul unor câmpuri statice ale clasei **`System`**:
  - **`System.in`** – fluxul de intrare pentru citire date (de tip **`java.io.InputStream`**)
  - **`System.out`** – fluxul de ieșire pentru afișare (de tip **`java.io.PrintStream`**)
- Afișare la consolă – se utilizează metodele clasei **`PrintStream`**:
  - **`println(valoare)`** – valoarea poate fi un tip de bază (**`int`**, **`char`**, **`double`**, ...) sau un obiect (caz în care se utilizează metoda **`toString`**)
  - **`printf(stringFormatare, valori)`** – similar cu funcția **`printf`** din C
- Documentația completă pentru formatare: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Formatter.html

- Exemplu:

```
System.out.println(“Salut!");                    // afișează Salut!
System.out.println(111);                         // afișează 111
System.out.printf("%d + %d = %d", 11, 22, 33); // afișează 11 + 22 = 33
```

Citire de la consolă

- Pentru citire de la consolă se utilizează clasa **`java.util.Scanner`** care conține:
  - Constructor care primește ca parametru un obiect de tip **`InputStream`** (**`System.in`** pentru consolă)
  - Metode de forma **`tip nextTip()`** și **`boolean hasNextTip()`** pentru citire și determinare existență, unde tip este un tip de bază (**`int`**, **`boolean`**, **...**)
  - Metodă **`String nextLine()`** pentru citire șiruri de caractere până la sfârșitul liniei (inclusiv spații)
  - Metodă **`String next()`** pentru citire șiruri de caractere până la următorul separator
  - Metodă **`setDelimiter(String)`** pentru setarea delimitatorului (implicit spații)

Exemplu:

```
var scanner = new Scanner(System.in);
scanner.useDelimiter("[\\s,\\,]+");          // acceptăm spații sau virgulă ca delimitatori
int cod = scanner.nextInt();                 // introducem 3, Maria
String nume = scanner.next();
System.out.printf("#%d - %s", cod, nume);    // afișează #3 - Maria
```

Variabile și Tipuri de date

- Declarare variabile
  - Similar cu declarațiile din C:

```
int i = 11;
double k;
```

  - Detecție automată a tipului folosind cuvântul cheie **`var`** (începând cu JDK 10):

```
var scanner = new Scanner(System.in);
```

- Declarare constante
  - Folosesc cuvântul cheie **`final`**:

```
final int NUMAR_MAXIM_ELEMENTE = 10;
```

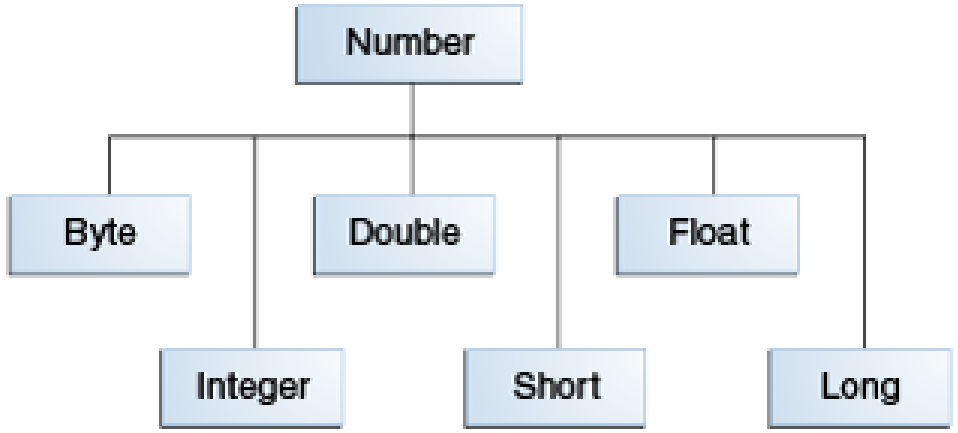
- Variabilele și constantele pot conține:
  - O valoare corespunzătoare unui tip de bază (int, double, ...)
  - O referință la un obiect

Tipuri Primitive

Keyword	Dimensiune	Clasa	Literali
boolean	1 bit	Boolean	***false***, *true*
Tipuri numerice întregi			
char	16 bits	Character	**'\u0000'**, 'a', '\u0041', '\\', '\'', '\n', 'ß'
byte	8 bits	Byte	-
short	16 bits	Short	-
int	32 bits	Integer	**0**, -12, 19, 2562234
long	64 bits	Long	**0L**, -12L, 19L, 922337203685477580**7**L
Tipuri numerice în virgulă mobilă			
float	32 bits	Float	**0.0f**, 1.23e100f, -1.23e-100f, .3f, 3.14F
double	64 bits	Double	**0.0d**, 1.3, 1.23456e300d, -1.23456e-300d, 1e1d

Clase *wrapper*. *Autoboxing* și *unboxing*

- Utilitate:
  - Conțin metode de conversie între tipurile primitive și între *String* și tipurile primitive
  - Permit utilizarea tipurilor de bază în situații în care este necesară o variabilă de tip referință (în special în cazul colecțiilor)
  - Furnizează constante utile (MIN_VALUE, MAX_VALUE)
- Conversii automate:
  - Autoboxing: tip primitiv → obiect wrapper
  - Unboxing: obiect wrapper → tip primitiv




```
int val = Integer.parseInt("11"); // conversie din String
Integer i = val;                  // Autoboxing
String test = i.toString();       // utilizare metodă din clasa Integer
int temp = i;                     // Unboxing
```

Operatori pentru tipuri primitive

- Operatorii pentru tipurile primitive sunt similari cu operatorii din C++

  Exemple:

```
boolean areProiect = true, prezentaExamen = true;
int notaExamen = 9;
boolean estePromovat = areProiect && prezentExamen && notaExamen >= 5;

long numarSecunde = 4451752342L;
int numarOre = (int)(numarSecunde / 60 / 60);

int varsta = 22;
varsta++;

double pi = 4.0 / 1 - 4.0 / 3 + 4.0 / 5 - 4.0 / 7 + 4.0 / 9;
```

Structuri de control

- Structurile de control sunt similare cu cele din C++

- ***if / else*** – instrucțiune alternativă simplă
- ***switch / case / default / break*** – instrucțiune alternativă multiplă

- ***for / while / do*** – instrucțiuni repetitive
- ***break / continue*** – instrucțiuni pentru întreruperea execuției instrucțiunilor repetitive

Șiruri de caractere

- Sunt manipulate prin intermediul obiectelor clasei **`String`** care:
  - Conțin o colecție de caractere
  - Sunt imutabile
- Construirea de obiecte de tip **`String`**:

```
String string1 = new String();                  // construire șir vid
String string2 = new String("Sir existent");    // construire pe baza de șir existent
String string3 = "Tot existent";                // echivalent cu declarația anterioară
```

- Determinarea dimensiunii și accesarea caracterelor:

```
String test = "Test";
for (int i = 0; i < test.length(); i++) {  // determinare număr caractere - .length()
    System.out.println(test.charAt(i));    // accesare caracter individual - .charAt(index)
}
```

Șiruri de caractere

- Concatenare și formatare:

```
// 1. Utilizare operator de concatenare, Buna și Java sunt variabile declarate anterior
String rezultat1 = Buna + " " + Java + "!";

// 2. Utilizare funcție statică format (sintaxa la fel ca la printf)
String rezultat2 = String.format("%s %s!", Buna, Java);
```

- Funcții de comparare și căutare:
  - `boolean` **`equals`**`(string s1)`: întoarce *true* dacă șirul curent este egal cu *s1*
    - Variantă *case insensitive:* `boolean` **`equalsIgnoreCase`**`(string s1)`
  - `int` **`compareTo`**`(string s1)`: întoarce *zero* dacă șirul curent este egal cu *s1*, valoare negativă dacă este mai mic, sau valoare pozitivă dacă este mai mare
    - Variantă *case insensitive:* `int` **`compareToIgnoreCase`**`(string s1)`
  - `int` **`indexOf`**`(string s1)`: întoarce poziția (=> 0) șirului *s1* în șirul curent sau -1 dacă *s1* nu a fost găsit
  - `boolean` **`startsWith`**`(string s1)`: întoarce *true* dacă șirul curent începe cu *s1*
    - similar `boolean` **`endsWith`**`(string s1)`

Șiruri de caractere

- Segmentare șiruri de caractere - `String[] split(String regex)`

```
String lista = "11,6,12,4,22,15,2";
String[] valoriLista = lista.split(",");
int[] valori = new int[valoriLista.length];
for (int index = 0; index < valori.length; index++) {
    valori[index] = Integer.parseInt(valoriLista[index]);
}
```

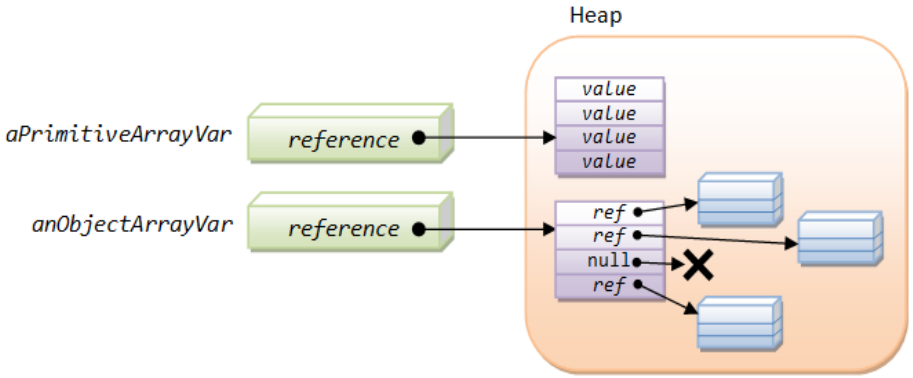
- Extragere subșir - `String substring(int begin, int end)`

```
String data = "20210226"; // yyyymmdd
int zi = Integer.parseInt(data.substring(6, 8));
```

- Metode de transformare:
  - `String toUpperCase()` – transformare litere mici în litere mari
  - `String toLowerCase()` – transformare litere mari în litere mici
  - `String trim()` – eliminare spații de la începutul și sfârșitul șirului

Masive unidimensionale - Declarare

- Masivele unidimensionale (vector) sunt obiecte care:
  - Conțin elemente de același tip (primitive sau referințe)
  - Au o dimensiune fixă specificată la inițializare



- Declarare și inițializare:

```
Student[] vStudenti;          // declarare fără inițializare
int[] vector1 = new int[3]; // declarare și inițializare cu valori default
int[] vector2 = {1, 2, 4};  // declarare și inițializare cu literal
```

- Accesare elemente, accesare dimensiune vector și parcurgere secvențială:

```
int n = vector1.length;
int val = vector1[1];

for (int element : vector1) { // foreach
      /* utilizare element */
}
```

Masive unidimensionale

- **Metode statice utile –** în clasa ***java.util.Arrays***:
  - `int binarySearch(vector, valoare)` – căutare element (-1 dacă elementul nu este găsit)
  - `void sort(vector)` – sortare elemente vector
  - `boolean equals(vector1, vector2)` – verifică dacă doi vectori sunt egali
- **Funcții cu număr variabil de argumente –** ***tip... denumire***
  - Au un singur parametru de acest fel, obligatoriu pe ultima poziție
  - Parametrul este tratat ca un vector în interiorul funcției

```
static int suma(int... valori) {
    int suma = 0;
    for (int element : valori) {
        suma += element;
    }
    return  suma;
}
// Apeluri: suma(1,5,2); sau int[] v = {1,2,3}; suma(v)
```

Masive multidimensionale

- Pot fi declarate massive multidimensionale - comportament similar cu *vector de vector*

  Exemplu:

```
int[][][] cub = {
        {{1, 2}, {3, 4}, {5, 6}},
        {{1, 2}, {3, 4}, {5, 6}}
         };
// o dimensiune a cubului este o matrice
int[][] m = cub[0]; // {{1, 2}, {3, 4}, {5, 6}}
int n = cub.length; // 2
// parcurgere cub ca vector de matrice
for (int[][] matrice : cub) {
    for (int[] vector : matrice) {
        for (int element : vector) {
            System.out.println(element);
        }
    }
}
```