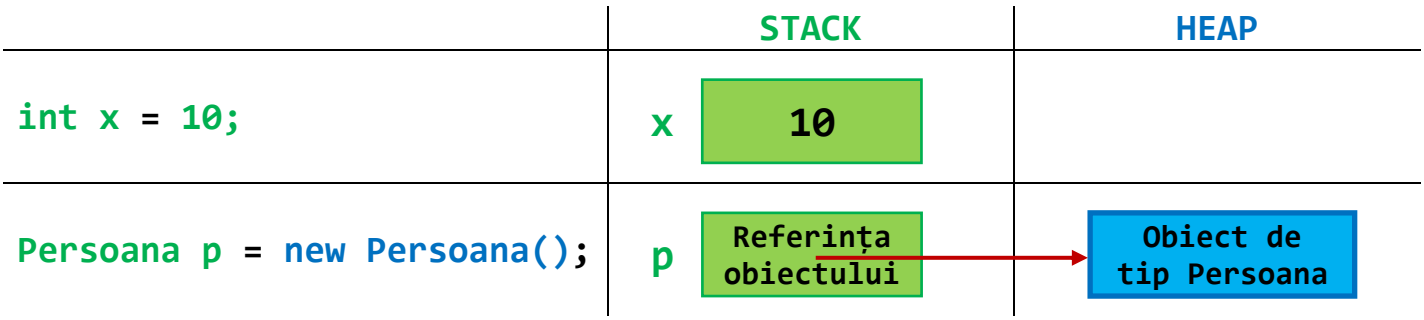
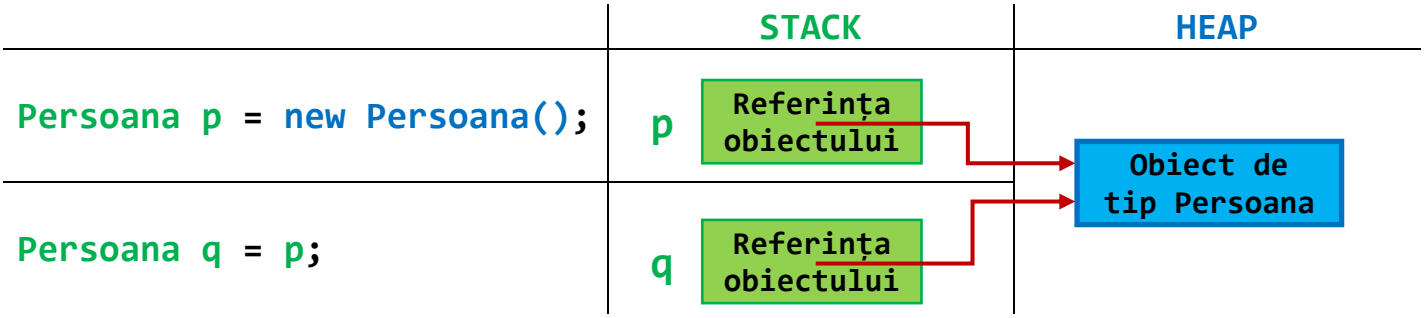
REFERINȚE

O **referință** reprezintă o modalitate de accesare a unei zone de memorie alocată dinamic (i.e., în zona de heap) prin intermediul adresei sale. Deși o referință utilizează adresa de memorie a unui obiect pentru a-l accesa, acest lucru nu se realizează într-un mod transparent pentru programator, respectiv acesta nu poate determina adresa respectivă și nici nu poate executa operații specifice pointerilor din limbajele C/C++!

Referințele, la fel ca variabilele de tip primitiv, sunt memorate în zona de stivă (stack). Practic, diferența dintre o variabilă de tip primitiv și una de tip referință constă în faptul că o variabilă de tip primitiv conține direct o valoare de un anumit tip, în timp ce o variabilă de tip referință conține doar "adresa" din zona de heap la care se află alocat un anumit obiect:



Efectuarea unei operații de atribuire între două referințe NU va conduce la realizarea unei copii a obiectului referit, ci obiectul respectiv va putea fi accesat prin intermediul ambelor referințe, ceea ce poate conduce la efecte colaterale nedorite:



Literalul `null` este utilizat pentru a indica faptul că o referință nu este asociată cu niciun obiect.

TABLOURI

În limbajul Java, tablourile sunt considerate variabile de tip referință, deci ele trebuie inițializate sau alocate dinamic înainte de a fi utilizate.

Declararea tablourilor unidimensionale (de fapt, a unor referințe spre tablouri unidimensionale!) se poate realiza în două moduri:

a) `tip_de_date[] tablou_1, tablou_2, ..., tablou_n;`
De exemplu, prin **`int[] a, b;`** se vor declara două referințe **a** și **b** spre tablouri unidimensionale cu elemente de tip **`int`** care trebuie ulterior să fie alocate dinamic!

b) `tip_de_date tablou_1[], tablou_2[], ..., tablou_n[];`
De exemplu, prin **`int a[], b;`** doar variabila **a** este o referință spre un tablou unidimensional cu elemente de tip **`int`**, iar variabila **b** este una de tipul primitiv **`int`**!

Tablourile unidimensionale pot fi inițializate în momentul declarării lor folosind un șir de valori, astfel:

a) `tip_de_date[] tablou = {valoare_1, ..., valoare_n};`
Exemplu: `int[] a = {1, 2, 3, 4, 5}, b = {10, 20, 30};`

b) `tip_de_date tablou[] = {valoare_1, ..., valoare_n};`
Exemplu: `int a[] = {1, 2, 3, 4, 5}, b[] = {10, 20, 30};`

Alocarea dinamică a tablourilor unidimensionale se realizează folosind operatorul **new**, astfel:

`tablou = new tip_de_date[număr_elemente];`

Exemple:

```
int a[];
.............
a = new int[5];
```

```
int[] a = null;
.............
a = new int[5];
```

```
int a[] = new int[5];
int []b = new int[7];
```

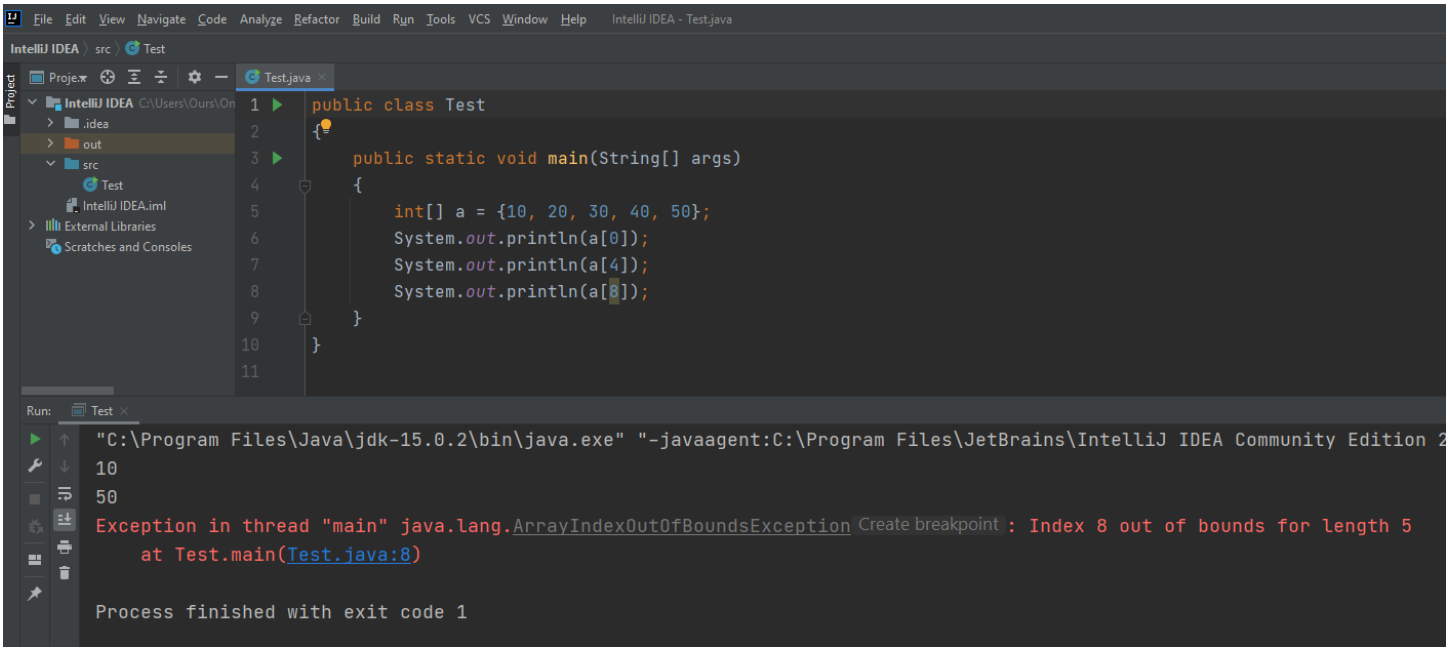
Toate elementele unui tablou alocat dinamic vor fi inițializate cu valori nule de tip!

După inițializarea sau alocarea dinamică a unui tablou, data membră **length** va conține dimensiunea fizică a tabloului, adică numărul maxim de elemente pe care le poate conține tabloul.

Parcurgerea unui tablou se poate realiza fie pozițional (prin intermediul indicilor elementelor), fie folosind o instrucțiune repetitivă de tipul **enhanced-for:**

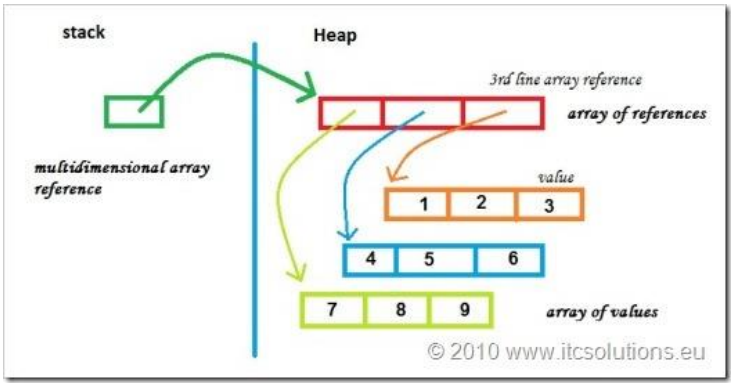
```
int[] a = {10, 20, 30, 40, 50};
System.out.println(a.length);          //se va afișa valoarea 5
for(int i = 0; i < a.length; i++)      //se vor afișa valorile 10 20 30 40 50
     System.out.print(a[i] + " ");

System.out.println();

int b[] = new int[7];
System.out.println(b.length);     //se va afișa valoarea 7
for(int elem : b)                 //se vor afișa valorile 0 0 0 0 0 0 0
     System.out.print(elem + " ");
```

Accesarea unui element dintr-un tablou al cărui indice este invalid (i.e., nu este cuprins între 0 și `tablou.length-1`) va duce la lansarea excepției `ArrayIndexOutOfBoundsException` în momentul rulării programului respectiv:



Observați faptul că excepția `ArrayIndexOutOfBoundsException` a fost lansată abia în momentul în care s-a încercat accesarea elementului `a[8]`, valorile elementelor `a[0]` și `a[4]` fiind afișate corect!

În limbajul Java tablourile bidimensionale sunt, de fapt, tablouri unidimensionale ale căror elemente sunt referințe spre tablouri unidimensionale, fiecare reprezentând câte o linie (sursa imaginii: Tutorial Java 6 - #4.3 Matrixes and Multidimensional Arrays | IT&C Solutions):



Declararea tablourilor bidimensionale (de fapt, a unor referințe spre tablouri bidimensionale!) se poate realiza în mai multe moduri:

a) `tip_de_date[][] tablou_1, tablou_2, ..., tablou_n;`

b) `tip_de_date tablou_1[][], tablou_2[][], ..., tablou_n[][];`

c) `tip_de_date[] tablou_1[], tablou_2[], ..., tablou_n[];`

Tablourile bidimensionale pot fi inițializate în momentul declarării, linie cu linie, așa cum se poate observa în exemplele de mai jos:

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

Liniile unui tablou bidimensional pot să aibă lungimi diferite:

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

Alocare dinamică a unui tablou bidimensional având liniile de aceeași lungime se realizează astfel:

referință_tablou = new tip_de_date[număr_linii][număr_coloane];

Exemple:

```
int[][] a = new int[5][3];
int a[][] = new int[7][7];
```

Alocare dinamică a unui tablou bidimensional având liniile de lungimi diferite se realizează astfel:

- se alocă un tablou bidimensional precizând doar numărul de linii:

tablou = new tip_de_date[număr_linii][];

- se alocă, pe rând, fiecare linie din tabloul bidimensional, precizând numărul de coloane:

```
tablou[0] = new tip_de_date[număr_coloane];
tablou[1] = new tip_de_date[număr_coloane];
.....................................
tablou[tablou.length-1] = new tip_de_date[număr_coloane];
```

Exemplu:

```
int[][] a = new int[3][];
a[0] = new int[7];
a[1] = new int[2];
a[2] = new int[5];
```

Parcurgerea unui tablou se poate realiza fie pozițional (prin intermediul indicilor elementelor), fie folosind o instrucțiune repetitivă de tipul **enhanced-for**.

Exemplu:

```
int[][] a = {{1, 2}, {3}, {4, 5, 6, 7}, {8, 9}};
for(int i = 0; i < a.length; i++)                    // se va afișa (de două ori):
{                                                    // 1 2
     for(int j = 0; j < a[i].length; j++)            // 3
          System.out.print(a[i][j] + " ");           // 4 5 6 7
     System.out.println();                           // 8 9
}
```

```
for(int[] linie : a)
{
     for(int elem : linie)
          System.out.print(elem + " ");
     System.out.println();
}
```

CLASA Arrays

Clasa `Arrays` este o clasă utilitară (i.e., conține doar metode statice) dedicată manipulării tablourilor. În continuare, vom prezenta mai multe metode ale acestei clase, din versiunea sa existentă începând cu Java 11:

- **`static String toString(Tip[] tablou)`** – furnizează o reprezentare a tabloului transmis ca parametru sub forma unui șir de caractere sau șirul "`null`" dacă referința sa este `null`. Elementele tabloului vor fi enumerate între o pereche de paranteze drepte și despărțite între ele prin câte o virgulă și un spațiu.

  Exemple:
  ```
  int[] t = {1, 2, 3, 4, 5};
  int[][] a = {{1, 2}, {3}, {4, 5, 6}};

  System.out.print(t);                    //[I@4dd8dc3
  System.out.print(Arrays.toString(t)); //[1, 2, 3, 4, 5]
  System.out.print(Arrays.toString(a)); //[[I@6d03e736, [I@568db2f2, [I@378bf509]
  ```

  Observați faptul că pentru tablourile bidimensionale se vor furniza referințele tablourilor corespunzătoare liniilor sale!

- **`static String deepToString(Tip[] tablou)`** – furnizează o reprezentare în adâncime a tabloului transmis ca parametru, respectiv dacă elementele tabloului sunt alte tablouri, atunci și acestea vor fi convertite în formatul precizat mai sus.

  Exemple:
  ```
  int[] t = {1, 2, 3, 4, 5};
  int[][] a = {{1, 2}, {3}, {4, 5, 6}};

  System.out.print(a);                    // [[I@4dd8dc3
  System.out.print(Arrays.toString(a)); //[[I@6d03e736, [I@568db2f2, [I@378bf509]
  System.out.print(Arrays.deepToString(a)); //[[1, 2], [3], [4, 5, 6]]
  ```

- **`static void fill(Tip[] tablou, Tip valoare)`** – atribuie tuturor elementelor tabloului unidimensional valoarea indicată.

  Exemplu:
  ```
  int[] t = {1, 2, 3, 4, 5};
  Arrays.fill(t, 7);
  System.out.println(Arrays.toString(t));        //[7, 7, 7, 7, 7]
  ```

- **`static boolean equals(Tip[] a, Tip[] b)`** – verifică dacă tablourile `a` și `b` sunt egale, respectiv dacă au același număr de elemente și elementele aflate pe aceleași poziții sunt egale.

  Exemple:
  ```
  int[] t = {1, 2, 3, 4, 5};
  int[] v = {1, 2, 3, 4, 5};
  int[] w = {1, 2, 4, 4, 5};
  System.out.println(Arrays.equals(t, v));       //true
  System.out.println(Arrays.equals(w, v));       //false

  int[][] a = {{1, 2}, {3}, {4, 5, 6}};
  int[][] b = {{1, 2}, {3}, {4, 5, 6}};
  System.out.println(Arrays.equals(a, b));       //false
  ```

  Observați faptul că în cazul tablourilor bidimensionale `a` și `b` se va afișa `false`, deoarece se vor compara referințele liniilor lor!

- **`static boolean deepEquals(Tip[] a, Tip[] b)`** – verifică în profunzime dacă tablourile `a` și `b` sunt egale, respectiv dacă elementele lor sunt tot tablouri le verifică egalitatea din punct de vedere al lungimilor și al elementelor aflate pe aceleași poziții, ci nu al referințelor.

  Exemplu:
  ```
  int[][] a = {{1, 2}, {3}, {4, 5, 6}};
  int[][] b = {{1, 2}, {3}, {4, 5, 6}};
  System.out.println(Arrays.deepEquals(a, b));   //true
  ```

- **`static int mismatch(Tip[] a, Tip[] b)`** – returnează cel mai mic indice `k` pentru care `a[k]` ≠ `b[k]` sau `-1` dacă tablourile sunt egale. Dacă tablourile `a` și `b` nu sunt egale, atunci valoarea indicelui `k` este cuprinsă între `0` și minimul dintre lungimile celor două tablouri `a` și `b`.

Exemple:

```
int[] t = {1, 2, 3};
int[] u = {1, 2, 3};
int[] v = {1, 2, 3, 4, 5};
int[] w = {1, 2, 4, 5, 5};

System.out.println(Arrays.mismatch(t, u)); //-1
System.out.println(Arrays.mismatch(u, v)); //3
System.out.println(Arrays.mismatch(w, v)); //2
```

Observați faptul că `Arrays.mismatch(u, v)` returnează 3, deoarece `u[0]==v[0]`, `u[1]==v[1]` și `u[2]==v[2]`, dar în tabloul v există, în plus față de tabloul u, elementele 4 și 5, deci indicele 3 se referă strict la tabloul v!

- **`static int compare(Tip[] a, Tip[] b)`** – compară lexicografic cele două tablouri și returnează următoarele valori:
  - o valoare negativă dacă tabloul a este mai mic în sens lexicografic decât tabloul b;
  - o valoare pozitivă dacă tabloul a este mai mare în sens lexicografic decât tabloul b;
  - valoarea `0` dacă tabloul a este egal în sens lexicografic cu tabloul b.

Exemple:

```
int[] t = {1, 2, 3, 4, 5};
int[] u = {1, 2, 3};
int[] v = {1, 2, 3, 4, 5};
int[] w = {1, 2, 4, 4, 5};

System.out.println(Arrays.compare(t, v)); // 0
System.out.println(Arrays.compare(t, u)); // > 0
System.out.println(Arrays.compare(w, v)); // > 0, deoarece 4 = w[3] > v[3] = 3
System.out.println(Arrays.compare(v, w)); // < 0, deoarece 3 = v[3] < w[3] = 4
```

- **`static Tip[] copyOf(Tip[] tablou, int nr_elem)`** – returnează un tablou format din primele `nr_elem` elemente ale tabloului dat ca parametru. Dacă `nr_elem` este strict mai mare decât lungimea tabloului, atunci se vor adăuga elemente nule de tip.

Exemple:

```
int[] t = {1, 2, 3, 4, 5};
int[] u = Arrays.copyOf(t, t.length);
int[] v = Arrays.copyOf(t, 3);
int[] w = Arrays.copyOf(t, 8);

System.out.println(Arrays.toString(u));  // [1, 2, 3, 4, 5]
System.out.println(Arrays.toString(v));  // [1, 2, 3]
System.out.println(Arrays.toString(w));  // [1, 2, 3, 4, 5, 0, 0, 0]
```

- **`static void sort(Tip[] tablou)`** – sortează crescător elementele tabloului dat ca parametru.

  Exemplu:
  ```
  int[] t = {2, 1, 5, 2, 5, 3, -10, 7, 4, 5};

  Arrays.sort(t);
  System.out.println(Arrays.toString(t)); // [-10, 1, 2, 2, 3, 4, 5, 5, 5, 7]
  ```

- **`static int binarySearch(Tip[] tablou, Tip valoare)`** – caută valoarea indicată în tabloul dat folosind algoritmul de căutare binară. Tabloul trebuie să fie sortat crescător, deoarece, în caz contrar, rezultatul furnizat nu va fi corect! Metoda returnează indicele pe care se găsește valoarea respectivă în tablou (dacă valoarea se găsește de mai multe ori în tablou, atunci se returna unul dintre indicii corespunzători) sau, dacă valoarea nu există în tablou, o valoare negativă p cu proprietatea că `-p-1` reprezintă indicele unde ar putea fi inserată valoarea căutată în tablou astfel încât acesta să rămână sortat crescător (i.e., indicele unde ar fi trebuit să se găsească valoarea căutată).

  Exemple:
  ```
  int[] t = {1, 2, 7, 7, 10, 10, 10, 21};

  System.out.println(Arrays.binarySearch(t, 10)); // 5

  System.out.println(Arrays.binarySearch(t, 9));  // -5, deoarece valoarea 9 ar
                        // trebui să se găsească în tablou pe poziția -(-5)-1 = 4

  System.out.println(Arrays.binarySearch(t, -9)); // -1, deoarece valoarea -19 ar
                        // trebui să se găsească în tablou pe poziția -(-1)-1 = 0

  System.out.println(Arrays.binarySearch(t, 99)); // -9, deoarece valoarea 99 ar
                        // trebui să se găsească în tablou pe poziția -(-9)-1 = 8
  ```

În afara metodelor prezentate mai sus, în clasa `Arrays` mai sunt definite și alte metode, majoritatea fiind variante ale celor prezentate mai sus (de exemplu, metode care nu prelucrează un tablou întreg, ci doar o secvență a sa, precizată prin 2 indici): https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html.