Primitivni tipi

`boolean`	1-bit truth values	`true`, `false`
`char`	16-bit characters	`'a'`, `'A'`, `'0'`, `'#'`, ...
`byte`	8-bit integers	..., -1, 0, 1, ..., 127
`short`	16-bit integers	..., -1, 0, 1, ..., 32767
`int`	32-bit integers	..., -1, 0, 1, ...
`long`	64-bit integers	..., -1, 0, 1, ...
`float`	32-bit floating point numbers	...-3.14 ...0.0 ...3.14 ...
`double`	64-bit floating point numbers	...-3.14 ...0.0 ...3.14 ...

Referenčni tipi

Ostali tipi v Javi so referenčni tipi (reference types)

Vrednost referenčnega tipa je:

- ali `null`
- ali referenca na objektu (object) ali na tabeli (array). (Tabela je tudi objekt.)

Tabelni tipi

Če je `t` tip, je tip tudi

`t[]`

Tip `t[]` je referenčni tip.

- `null` je možna vrednost.
- Druge vrednosti so reference tabele z elementi tipa `t`.

```
int[] nums;
```

Spremenljivka `nums` je deklarirana kot tabela tipa `int[]`. Vrednosti še nima.

```
nums = null;
nums = new int[3];
```

Zdaj je `nums` tabela s tremi elementi: `nums[0]`, `nums[1]` in `nums[2]`. Vsak ima začetno vrednost `0`.

```
Arrays.fill(nums, 7);
```

Izpolni `nums` z vrednostjo `7`: `nums[0]=7`, `nums[1]=7` in `nums[2]=7`. Potrebno je `import java.util.Arrays;`

```
Random r = new Random();
for (i=0; i<3; i++) nums[i] = r.nextInt(6);
```

Izpolni `nums` z naključnimi vrednostmi od `0` do `5`. Potrebno je `import java.util.Random;`

Lahko definiramo tabelo istočasno kot jo deklariramo.

```
int[] nums = new int[3];
```

Spremenljivka `nums` je deklarirana kot tabela tipa `int[]`. Vrednost je referenca tabele s tremi elementi: `nums[0]`, `nums[1]` in `nums[2]`. Vsak ima začetno vrednost `0`.

```
int[] nums = {2,3,5,7};
```

Spremenljivka `nums` je deklarirana kot tabela tipa `int[]`. Vrednost je referenca tabele s štirimi elementi: `nums[0]=2`, `nums[1]=3`, `nums[2]=5` in `nums[3]=7`.

```
public class QuickSort {

    public static Random r = new Random();

    public static float[] array;

    public static void main(String[] args) {

        final int SIZE = 7;
        //Create a randomly populated array of floats
        array = new float[SIZE];
        for (int i = 0; i<SIZE; i++) array[i] = r.nextFloat();

        System.out.println("Before sorting:");
        System.out.println(Arrays.toString(array));

        //Randomised quicksort the array
        rqs(0,SIZE-1);

        System.out.println("After sorting:");
        System.out.println(Arrays.toString(array));
    }

    // rqs method goes here //

    // partition method goes here//
}
```

```
// rqs(i,j) where i <= j+1
// randomised quicksort of region from index i to index j of array
// (if j=i-1 then the region being sorted is empty)

private static void rqs (int i, int j) {
    if (i < j) {// only sort if region has 2 or more elements

        //Choose a random pivot element
        float pval = array[i + r.nextInt(j-i+1)];

        int k = partition (i, pval, j);
        rqs (array, i, k-1);
        rqs (array, k+1, j);
    }
}
```

```
// partition(i,pval,j) where i <= k <= j
// uses the pivot value pval to partition array[i,...,j].
// Array entries are rearranged so that entries smaller than pval lie to
// its left and entries larger than pval lie to its right.
// The method returns the new index of pval in the rearranged array.

private static int partition (int i, float pval, int j) {
    while (i < j) {
        if (array[i] < pval) i++; //smaller elements stay on left
        else if (array[j] > pval) j--; //larger elements stay on right
        else if (array[i] == pval && array[j] == pval) j--;
        else { // array[i] and array[j] out of place so swap
            float temp = array[j];
            array[j] = array[i];
            array[i] = temp;
        }
    }
    return i;
}
```

```
double[][] aa = new double[3][5];
```

Spremenljivka `aa` je 2-dimenzionalna tabela z 3×5 vrednostmi tipa `double`. Vsak ima začetno vrednost `0.0`.

```
double d = 1/Math.sqrt(2);
double[][] rr = {{d,-d}, {d, d}};
```

`dd` je 2-dimenzionalna tabela ki reprezentira 2×2 matrika:

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

```
// multiply (l, aa, m, bb, n) performs matrix multiplication aa x bb
// where aa is an l x m matrix and bb is an m x n matrix

double[][] multiply (int l, double[][] aa, int m, double[][] bb, int n) {
    double[][] cc = new double[l][n];
    for (int i=0; i<l; i++) {
        for (int k=0; k<n; k++) {
            // at this point cc[i][k] is 0
            for (int j=0; j<m; j++) {
                cc[i][k] += aa[i][j] * bb[j][k];
            }
        }
    }
    return cc;
}
```

Datoteke

```
FileReader in = new FileReader("/Path/in.txt");
FileWriter out = new FileWriter("/Path/out.txt");
```

Odpri datoteki `in.txt` in `out.txt` za vzhod (input) in izhod (output).

```
while ((c = in.read()) != -1) out.write(c);
```

Kopira datoteko `in.txt` do `out.txt`. Vrednosti spremenljivke `c` so znaki iz datoteke `in.txt`. Ko je `c` enak `-1` pomeni konec datoteke (end of file) `in.txt`. Tip spremenljivke `c` je `int`.

```
out.close(); in.close();
```

Zapri datoteki .

Potrebno je `import java.io.*;`

Operacije na datotekah lakho vrnijo izjeme (exceptions) tipa `IOException`.

Kadar moramo ujeti (catch) izjeme, na splošno uporabljamo

`try` *`try-clause`* `catch` *`catch-clause`* `finally` *`finally-clause`*

Vsaj eden izmed `catch` *`catch-clause`* in `finally` *`finally-clause`* je obvezen.

Kadar se ukvarjamo z datotekami je posebna konstukcija try-with-resources bolj koristna.

Try-with-resources

```
try (FileReader in = new FileReader("/Path/in.txt");
     FileWriter out = new FileWriter("/Path/out.txt")) {
    int c;
    while ((c = in.read()) != -1) out.write(c);
} catch (IOException exn) {
    exn.printStackTrace();
}
```

Ta koda uporablja try-with-resources

```
try (resource-declarations) try-clause
   catch catch-clause finally finally-clause
```

- Niti `catch` *`catch-clause`* niti `finally` *`finally-clause`* ni obvezen.
- Deklarirani viri (resources) (na primeru `in` in `out`) se zaprejo avtomatično (na nasprotnem vrstnem redu kot red deklaracij)

```
// Reads a text file and outputs to a text file all consecutive
// sequences of letters as words separated by single spaces
try (FileReader in = new FileReader("/Path/in.txt");
     FileWriter out = new FileWriter("/Path/out.txt")) {
    int c;
    boolean space = false;
    while ((c = in.read()) != -1) {
        if (Character.isLetter(c)) {
            // if c is a letter write to output file
            // and flag that space is needed after
            out.write(c); space = true;
        }
        else if (space) {
            // if c is not a letter then write a
            // space only if space=true
            out.write((int) ' ');
            // After one space we do not need another
            space = false;
        }
    }
} catch (IOException exn) {
    exn.printStackTrace();
}
```