Увод в програмирането

с Java

Преговор – Въведение в Java

Съдържание

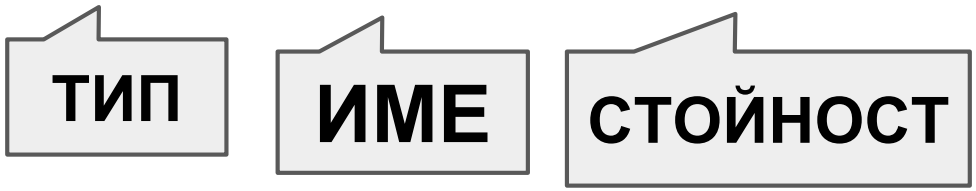
- Типове данни
- Оператори
- Изрази
- Условни конструкции
- Цикли
- Масиви
- Методи
- Символни низове

Променливи

Променливи

- Променливата е място, където съхраняваме частичка информация
- Нещо като чекмедже - вътре в чекмеджето има съдържание, а самото чекмедже си има име, може да съхранява точно определен тип информация

```
String myName = “Tihomir”;
```



Боравене с променливи

- Променливите трябва да бъдат създавани. Това става като се казва от какъв тип е променливата и какво й е името.

String password;

- В този случай създаваме променлива от тип String с име password. Не е задължително да й задаваме първоначална стойност, както на предния слайд

Боравене с променливи

След като веднъж е създадена една променлива ние можем да я използваме надолу в кода многократно.

```
String password;

password = “123456”;

System.out.print(password);

// Показва 123456 в конзолата

password = “qwerty”;

System.out.print(password);

// Показва qwerty в конзолата
```

Типове данни

Типове данни – цели числа

- byte

От -128 до 127. Заема 8 бита памет.

- int

От -2 147 483 648 до 2 147 483 647. Заема 32 бита памет.

- long

От -9,223,372,036,854,775,808

до 9,223,372,036,854,775,807. Заема 64 бита памет.

Типове данни – дробни числа

- float

Заема 32 бита памет. Има прецизност 7 знака след десетичната запетая.

- double

Заема 64 бита памет. Има прецизност 13 знака след запетаята.

Не са точни и не трябва да се използват при сметки с валута.

Типове данни – знаци

- char

Може да държи един знак

- String

Може да съхранява неограничено количество текст

Типове данни – булеви

- boolean

Може да бъде само две стойности - true или false.

```
boolean isRaining = true;

boolean hasError = false;
```

Пример

```
int year = 2015;

byte age = 18;

String name = “Pesho”;

long money = 5 555 555 555;

char firstLetter = ‘P’;

boolean isMale = true;

double height = 1.82;
```

ОПЕРАТОРИ

ОПЕРАТОРИ

Операторите са специални символи, които извършват действие върху един, два или три *операнда* и връщат резултат.

Операнд е променливата, чиято стойност се използва/променя от оператора.

Пример: a + b < c

“+” и “<” са *оператори* a, “b” и “c” са *операнди*

ТИПОВЕ ОПЕРАТОРИ

Операторите могат да се делят по различен начин:

- според броя променливи /операнди/, върху които се прилага
- според разположението на операнда спрямо оператора
- според резултата, който се получава от прилагането на оператора

Оператори според броя операнди

Брой операнда (аргументи)	Тип оператор	Пример
1	Унарен (unary)	а++
2	Бинарен (binary)	а + б
3	Тернарен (ternary)	а ? б : с

Оператори според разположението им спрямо операнда

Разположение	Тип оператор	Пример
Преди операнда	Префиксен	--expr
След операнда	Постфиксен	expr++

Унарни оператори

Оператори, които имат един операнд, се наричат унарни. Резултатът от изпълнението им е число, освен при + където може да е текст и при ! където може да е истина или лъжа.

Унарен Оператор	**Описание**
+	Унарен оператор плюс; индикация за положителна стойност (числата са позитивни и без да се пише +)
-	Унарен оператор минус; прави израз негативен
++	Оператор за инкрементиране; увеличава стойността на операнда с 1
--	Оператор за декрементиране; намалява стойността на операнда с 1
!	Логическо отрицание; сменя стойността на булев израз или променлива

Пример

```
int a = 5;
a++;
int b = 4;
b--;

System.out.println(“a=” + a);
System.out.println(“b=” + b);
```

++A или A++

Има разлика дали операторът ще е пред или след операнда, когато двете са част от по-голям израз.

Пример:

```
int a = 5;

System.out.print(a++); // 5
System.out.print(a);   // 6
System.out.print(++a); // 7
```

Аритметични оператори

Аритметични оператори

Оператори за извършване на аритметични операции. Резултатът от изпълнението им е число.

Аритметичен оператор	**Описание**
+	Оператор за събиране (използва се и за конкатенация на низове – ще го видите в следващите лекции)
-	Оператор за изваждане
*	Оператор за умножение
/	Оператор за деление
%	Деление по модул

Аритметични оператори

Когато се използва операторът за деление / с целочислен тип (integer), върнатият резултат е отново целочислен (без закръгляне). За да се вземе остатъкът от делене на цели числа се използва оператора %.

Пример

```
int a = 7;

int b = 4;

int div = a / b;

int mod = a % b;

float fDiv = a / b;

System.out.println(“div = ” + div);  //1

System.out.println(“mod = ” + mod);  //3

System.out.println(“fDiv = ” + fDiv);  //1,0
```

Пример

```
float a = 7;

float b = 4;

float div = a / b;

System.out.println("div = " + div); // 1.75
```

Пример

Колко ще се изведе на конзолата?

- System.out.println(7 / 2);
- System.out.println(7.4 / 2);

Оператори за сравнение

Оператори за сравнение

Дават възможност за сравнение на два операнда. Резултатът от изпълнението им е истина или лъжа.

Оператор за сравнение	Описание
==	Равно
!=	Различно
>	По-голямо
>=	По-голямо или равно
<	По-малко
<=	По-малко или равно

Пример

```
int a = 5;

int b = 6;

boolean greater = a > b;

boolean smaller = a < b;

boolean diff = a != b;

System.out.println(“a > b ->” + greater);

System.out.println(“a < b ->” + smaller);

System.out.println(“a != b ->” + diff);
```

Логически оператори

Логически оператори

Оператори за работа с булеви данни и булеви изрази.

Логически оператор	Описание
&&	Логическо "И"
||	Логическо "ИЛИ"
!	Логическо отрицание; сменя стойността на булев израз или променлива

Таблица на истинността (Truth table)

При логическото И (&&) всички операнди трябва да са истина за да бъде целия израз истина, при логическото ИЛИ (||) е достатъчно един операнд да е истина за да бъде целия израз истина.

x	y	!x	!y	x && y	x || y
true	true	false	false	true	true
true	false	false	true	false	true
false	true	true	false	false	true
false	false	true	true	false	false

Short-Circuiting

При логическите оператори се изпълнява т.нар. “short-circuiting”, т.е. стойността на втория операнд се оценява само, ако е необходимо. При оператор „И“, ако първият аргумент е false, вторият не се оценява и стойността на израза е false. При оператор „ИЛИ“, ако първият аргумент е true, вторият не се оценява и стойността на израза е true.

Пример

```
boolean a = true;

boolean b = false;

System.out.println(a || b); // true

System.out.println(a && b); // false
```

Оператори за присвояване

Оператори за присвояване

- Прост оператор за присвояване. Задава стойност на променлива. Вече сте го учили в предишната лекция.
- Комбинирани оператори за присвояване Позволяват съкратен запис на две операции.

Пример

```
String myName = „Lilly";

int a = 5;

a += 10; // същото като a = a + 10;

a -= 7; // същото като a = a - 7;

a *= 4; // същото като a = a * 4;

a /= 2; // същото като a = a / 2;

a %= 3; // същото като a = a % 3;

System.out.println("a = " + a);
```

Предимство на операторите

Предимство на операторите

Някои оператори имат приоритет над други. Операторите с по-висок приоритет се изчисляват преди тези с по-нисък. Операторът **()** служи за промяна на приоритета на операторите и се изчислява пръв, също както в математиката.

В таблицата са показани приоритетите на операторите в Java:

Оператор	Оператор	Предимство
Постфиксни	postfix	expr++ expr--
Унарни	unary	++expr --expr +expr -expr ~ !
За умножение	multiplicative	* / %
За събиране	additive	+ -
Побитово отместване	shift	<< >> >>>
За сравнение	relational	< > <= >= instanceof
За равенство	equality	== !=
Побитово „И“	bitwise AND	&
Побитово изключващо „ИЛИ“	bitwise exclusive OR	^
Побитово „ИЛИ“	bitwise inclusive OR	|
Логическо „И“	logical AND	&&
Логическо „ИЛИ“	logical OR	||
Тернарен	ternary	? :
За присвояване	assignment	=

* Операторите с по-малък и светъл шрифт няма да бъдат разглеждани в текущата лекция

Условен оператор

Връща различен резултат в зависимост от изпълнението на дадено условие.

Условие ? Стойност1 : Стойност2

Нарича се тернарен оператор, т.к. има 3 аргумента.

Пример

Дадени са 2 цели числа. Да се напише израз, който връща стойността на по-голямото число, умножена по 2.

Решение:

```
int a = 5;

int b = 7;

int doubledGreater = (a > b) ? 2 * a : 2 * b;

System.out.println(doubledGreater);
```

Четене от конзолата

Четене от конзолата

Може да го използвате за решаване на задачите.

```
Scanner input = new Scanner(System.in);

System.out.print("Enter number:");

int number = input.nextInt();
```

Изрази

Изрази

Израз е последователност от оператори, литерали и променливи, която връща някаква стойност.

Изразите имат тип (int, double, boolean ...) и стойност.

Пример:

```
// Изчисляване на лицето на кръг

double surface = Math.PI * r * r;
```

Пример

```
int x = (10 + 5)/2;   // Израз от тип Int

int y = (x + 2) * (x - 4) + (2 * x - 6)/2; // Израз от тип Int

boolean areOddNumbers = (x % 2 != 0) && (y % 2 == 1); // Израз от тип boolean
```

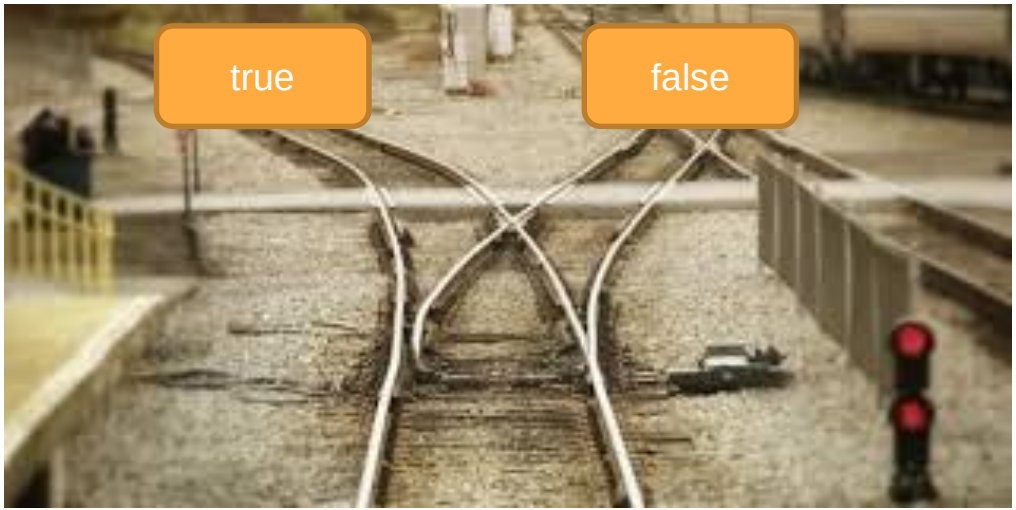
Задача

Иван тренира футбол. Иван е зает човек. Той тренира футбол само през уикенда. Понякога се прибира в Шумен. Когато е в Шумен той тренира футбол. Освен ако не е в Шумен през уикенда, тогава се среща с други приятели и се напиват. Напишете програма, която казва дали Иван ще играе футбол.

Условни конструкции

Условни конструкции

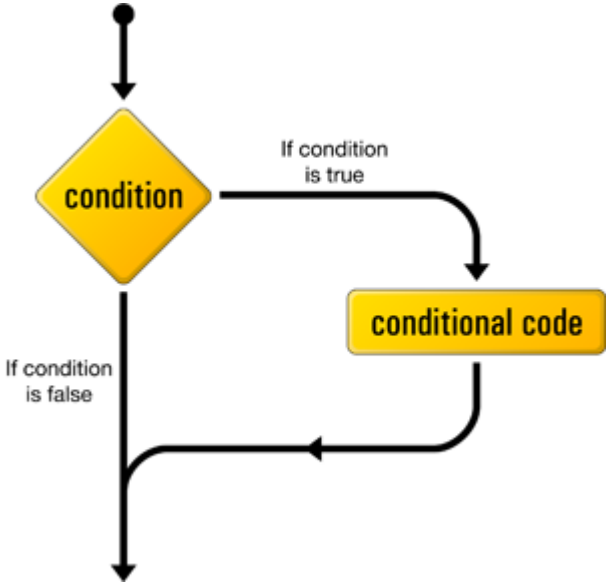
Позволяват изпълнението на код, само ако някакво условие е вярно.



Условна конструкция if

Позволява изпълнението на даден блок код само ако е изпълнено дадено условие.

```
if (условие) {

//код, който ще се изпълни

//ако условието е вярно

}

//код, който ще се изпълни винаги
```



Условна конструкция if-else

Изпълнява даден блок код при истинността на дадено условие и друг блок код, когато условието не е изпълнено.

```
if (условие) {
//код, който ще се изпълни
//ако условието е вярно
} else {
//код, който ще се изпълни
//ако условието е грешно
}
```

Условна конструкция switch-case

Конструкцията **switch** е един ясен начин за имплементиране на избор между множество варианти (тоест, избор между няколко различни пътища за изпълнение). Тя изисква селектор, който се изчислява до цяло число от типа **int**, **byte**,**char** или **enum**. Ако искаме да използваме, например, низ или число с плаваща запетая като селектор, това няма да работи в **switch** конструкция. За нецелочислени типове данни трябва да използваме последователност от **if** конструкции.

Условна конструкция switch-case

Прави избор от части на код за изпълнение, в зависимост от стойността на даден израз.

```
switch (селектор) {

case стойност-1 : конструкция; break;

case стойност-2 : конструкция; break;

case стойност-3 : конструкция; break;

…

  default: конструкция;

}
```

Пример

```
int number = 6;

switch (number) {

        case 1:

        case 10: System.out.println("Числото не е просто!"); break;

        case 2:

        case 5:

        case 7: System.out.println("Числото е просто!"); break;

        default: System.out.println("Не знам какво е това число!");

}
```

Задача

Имаме две числа. Изпечатайте сбора им, освен ако двете числа са равни, тогава изпечатайте сбора им по две.

(1, 2) → 3

(3, 2) → 5

(2, 2) → 8

Цикли

Какво наричаме цикъл?

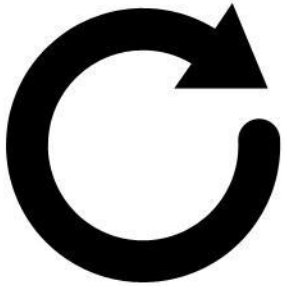
Понякога се налага да изпълняваме един и същи код многократно. За да не трябва повторно да пишем този код много пъти, в програмирането съществува концепцията за **цикъл (loop)** - повторено изпълнение на даден набор от операции. Всяко отделно изпълнение на операциите се нарича **итерация**. В Java съществуват три вида цикли - **for, while** и **do-while**.

Конструкция за цикъл while

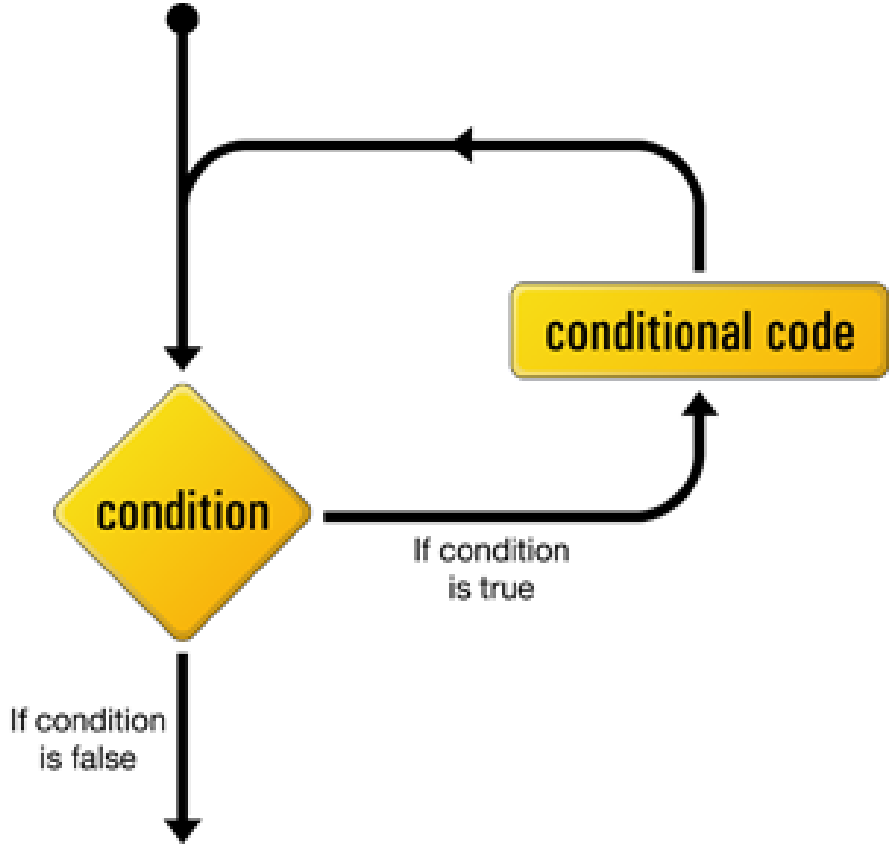
while изпълнява набор от операции, ***докато*** дадено условие е вярно:

```
while (условие) {
        израз1;
    израз2;
    ...
    изразn;
```

Конструкция за цикъл while



LOOPS REPEAT
ACTIONS...
SO YOU DON'T HAVE TO



Цикъл do-while

Конструкцията **do-while** е аналогична на **while**, като разликата е, че условието се оценява след изпълнението на операциите в цикъла (гарантираме **най-малко едно изпълнение**):

```
do {

statement1;

statement2;

…

}

while (Boolean expression);
```

Цикъл do-while - пример

```
int number = 1;

do {

System.out.println(number);

number++;

}

while (number <= 10);
```

Безкрайни цикли и специални оператори

Безкраен цикъл

Какво ще се случи при изпълнението на следния код?

```
while (true) {

System.out.println(“Blablabla”);

}
```

Безкраен цикъл

Безкрайният цикъл е цикъл, който никога не завършва. Създаването на безкраен цикъл трябва да се избягва, тъй като той кара програмата ни да “увисне” - да продължава да изпълнява едно и също нещо безкрайно, което от потребителска гледна точка не е желателно да става.

Оператор break

Операторът **break** се използва за **прекъсване на цикъл**. Всичко след него се пренебрегва и **програмата излиза от цикъла**:

```
while (условие) {

// код, който ще се изпълни

break;

// код, който няма да се изпълни

}
```

Оператор break - пример

```
while (true) {
System.out.println(“You’ll see me”);
break;
System.out.println(“But you won’t see me :(“);
}
System.out.println(“Out of loop”);
```

Оператор break - пример

```
Scanner input = new Scanner(System.in);

int number;

while (true){

        System.out.println("Please, insert a number: ");

                number = input.nextInt();

                if (number == 0)

                        break;

}
```

Оператор continue

Операторът **continue** се използва за преминаване към следващата **итерация**. Всичко след него се пренебрегва, но програмата не излиза от цикъла, а се **връща към оценяване на условието**:

```
while (условие) {

// код, който ще се изпълни

continue;

// код, който няма да се изпълни,

а вместо него, ще се оцени наново условието

}
```

Цикъл For

Използва се когато знаем колко точно завъртания ще има цикъла.

```
for(int i = 0; i < 10; i++) {

System.out.println(i);

}
```

i е брояча, който се променя с всяко завъртане на цикъла. Трябва да му зададем първоначална стойност, какво е условието при което цикъла ще спре и също така - как брояча ще се променя. Променливите, които се инициализират в рамките на цикъла не съществуват извън него.

Пример

Сумирайте числата от 1 до n:

```
int n = 20;

int sum = 0;

for (int i = 1; i <= n; i++) {

sum += i;

}

System.out.print(sum);
```

Вложени цикли

Вложени цикли

Вложените цикли представляват конструкция от няколко цикъла един в друг. Най-вътрешния цикъл се изпълнява най-много пъти.

```
for (initialization; test; update) {
    for (initialization; test; update) {
        statements;
    }
    …
}
```

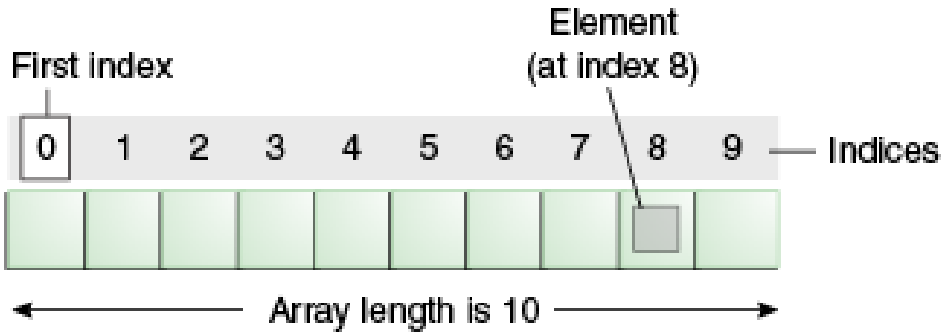
Задача

Напишете програма, която отпечатва на конзолата числата от 1 до N, които не се делят на 3 и 7. Числото N се чете от стандартния вход.

Масиви

Масив

Обект, който съдържа в себе си определен брой други обекти от един и същ тип.



Деклариране на масив

От всеки един тип може да се направи масив. Той се използва, когато имаме много променливи от един и същ тип и ни се налага да ги обхождаме машинно.

```
int[] numbers = {1, 2, 3, 4}; //масив от int

boolean[] isRaining = {true, true, false, false};
```

Други начини за създаване

Когато няма да задаваме стойността на елементите в началото, но знаем техния брой:

```
int[] bones = new int[206];

//сега създадохме масив от int с 206 елемента, който е празен - елементите му нямат стойност.
```

Достъп до елементите

Достъпът до елементите става с квадратни скоби:

String[] bones = new String[206];

bones[0] = “ulna”;

//първия елемент на масива има стойност, всички останали са празни

Накратко

Масивът:

- Е поредица от елементи от един и същи тип
- Има фиксиран размер (брой елементи)
- Подрежда елементите по фиксиран начин

Деклариране

Има два начина:

•Когато знаем кои са елементите:

–int[] myArray = {1, 2, 3, 4, 5};

•Когато знаем само броя им:

–int[] myArray = new int[5];

– Операторът “new” заделя памет за масива.

Достъпване

- Могат да се достъпват чрез []
- Индексът на първия елемент е 0
- Индексът на последния елемент е дължината на масива - 1

Обхождане

Често ни се налага да минем през всички елементи на масив и да направим нещо с тях. Този процес се казва обхождане.

Възможен е чрез цикли.

```
int[] array = {3 ,5 ,6, 6,7,8};

for (int i = 0; i < array.length; i++){

    array[i]=5+i;

}
```

Сортиране

Да сортираме един масив, означава да подредим елементите му според някакъв критерий.

Пример: Да наредим едно тесте от карти

Пример: Имаме масив от оценки на студенти и трябва да ги подредим по нарастване - от най-ниските към най-високите.

Пример: Имаме масив от имена на студенти и трябва да ги подредим по азбучен ред.

Алгоритми за сортиране

Алгоритмите за сортиране ни дават решение на задачата как да сортираме един масив от данни. Те са едни от основните в компютърните науки и се използват често, както за обучителни цели, така и в практиката.

Класифицираме ги според няколко критерия:

- стабилност
- бързина (времева сложност)
- използвана памет
- ефективност
- др.

Алгоритми за сортиране

Наивни:

- “метод на мехурчето” (Bubble Sort)
- сортиране чрез вмъкване (Insertion Sort)
- сортиране чрез пряка селекция (Selection Sort)

Ефективни:

- бърза сортировка (Quick Sort)
- сортиране чрез сливане (Merge Sort)
- сортиране чрез двоична пирамида (Heap Sort)

Bubble Sort (algorithm)

```
int[] array = { 6, 9, 3, 1, 8 };
int temp = 0;

for (int i = 0; i < array.length; i++) {
    for (int j = 1; j < (array.length - i); j++) {

        if (array[j - 1] > array[j]) {
            // swap the elements!
            temp = array[j - 1];
            array[j - 1] = array[j];
            array[j] = temp;
        }
    }
}
System.out.println(Arrays.toString(array));
```

Insertion Sort (algorithm)

```
int[] array = { 3, 9, 8, 1, 6 };
for (int i = 1; i < array.length; i++) {
int j = i;
int temp;
while (j > 0 && array[j - 1] > array[j]) {
temp = array[j - 1];
array[j - 1] = array[j];
array[j] = temp;
j = j - 1;
}
}
System.out.println(Arrays.toString(array));
```

Selection Sort (algorithm)

```
int[] array = { 3, 9, 2, 1, 6 };

for (int j = 0; j < array.length; j++) {
int minIndex = j;
for (int i = j + 1; i < array.length; i++) {
if (array[i] < array[minIndex]) {
minIndex = i;
}
}
int temp;
if (minIndex != j) {
temp = array[minIndex];
array[minIndex] = array[j];
array[j] = temp;
}
}
System.out.println(Arrays.toString(array));
```

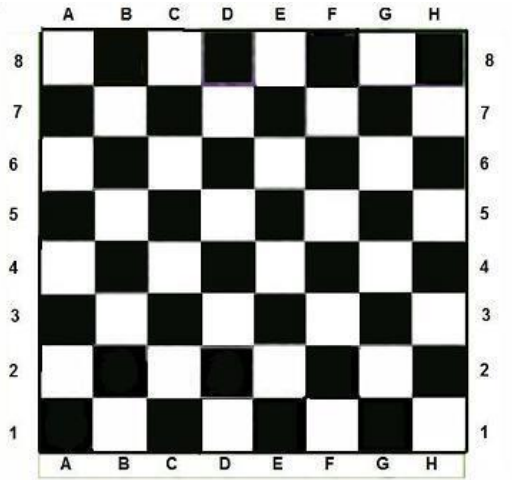
Задача

Сортирайте целочислен масив използвайки някой от изучените алгоритми за сортиране.

Многомерен масив

Масивите, с които се занимавахме досега, представят един ред обекти от някакъв тип. Често обаче ни се налага да представяме данните под формата на таблици (напр., таблица с оценки за всеки студент, в която всеки ред е даден студент, а всяка колона - оценка по даден предмет.).

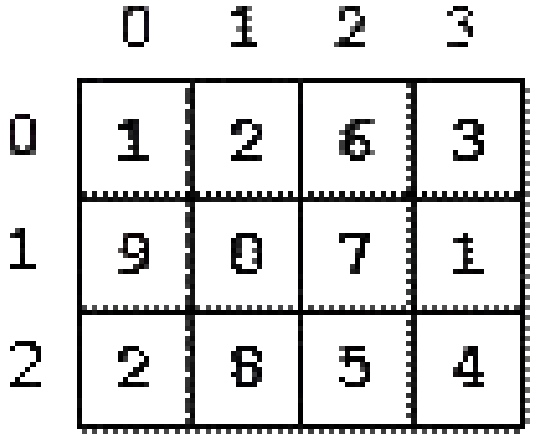
Id	Name	Email	Investments
231	Albert Master	albert.master@gmail.com	Bonds
210	Alfred Alan	aalan@gmail.com	Stocks
256	Alison Smart	asmart@biztalk.com	Residential Property
211	Ally Emery	allye@easymail.com	Stocks
248	Andrew Phips	andyp@mycorp.com	Stocks
234	Andy Mitchel	andym@hotmail.com	Stocks
226	Angus Robins	arobins@robins.com	Bonds
241	Ann Melan	ann_melan@iinet.com	Residential Property
225	Ben Bessel	benb@hotmail.com	Stocks
235	Bensen Romanolf	benr@albert.net	Bonds



Многомерен масив - декларация

Многомерните масиви са нищо друго, освен масив от масиви. Могат да имат n на брой измерения, но рядко в практиката се използват повече от 2.

```
int[][] twoDimentionalArray;

int[][][] threeDimentionalArray;

int[][] intMatrix = new int[3][4];
float[][] floatMatrix = new float[8][2];
String[][][] stringCube = new String[5][5][5];
```



Многомерен масив - инициализация

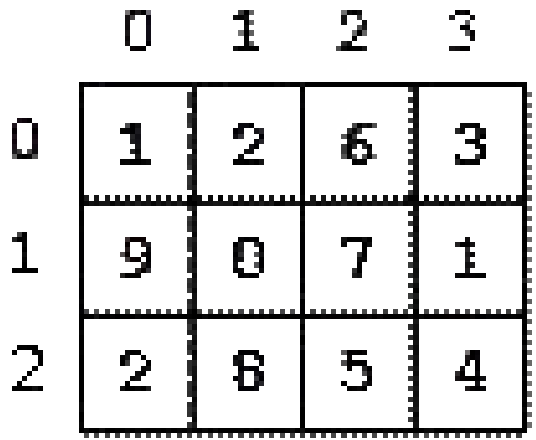
Инициализираме многомерните масиви по същия начин, както и едномерните.

```
int[][] matrix = {
    {1, 2, 3, 4}, // row 0 values
    {5, 6, 7, 8}, // row 1 values
};
```

Матрица

Матриците са двумерни масиви, т.е. масив от едномерни масиви. Имат редове и колони, като всеки ред е масив от елементи, а всяка колона - съвкупност от елементи с еднакъв индекс.

```
int[][] matrix = {
        {1, 2, 6, 3},
        {9, 0, 7, 1},
        {2, 8, 5, 4}
};
```



Достъп до елементите на многомерен масив

Както при едномерните масиви, можем да достъпваме елементите и на многомерен масив. За да вземем даден елемент, трябва да посочим номер на ред и номер на колона:

matrix[i][j];

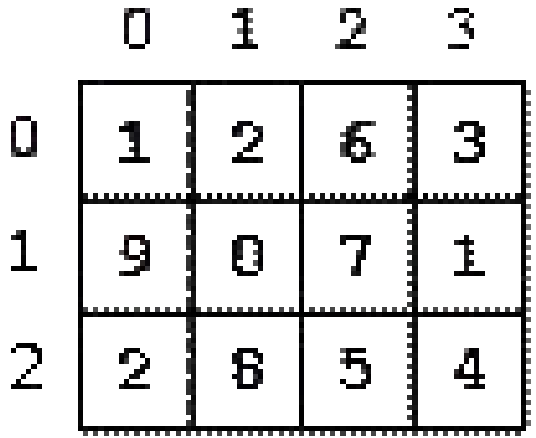
Например:

int element1 = matrix[0][1]; // element1 = 2

int element2 = matrix[2][2]; // element2 = 5

int element3 = matrix[1][2]; // element3 = ?

int element4 = matrix[3][0]; // element4 = ?



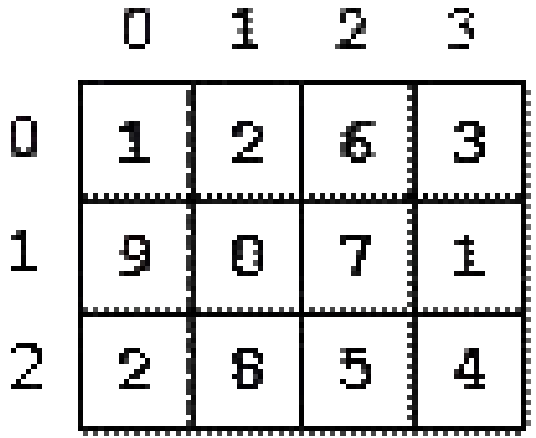
Размер на матрица

За да намерим броя на редовете на една матрица, използваме метода . length. Тъй като матрицата е просто масив от едномерни масиви, length ни дава размера на този масив:

```
int rows = matrix.length; // 3
```

За да намерим броя на колоните, прилагаме length върху някой от редовете, например:

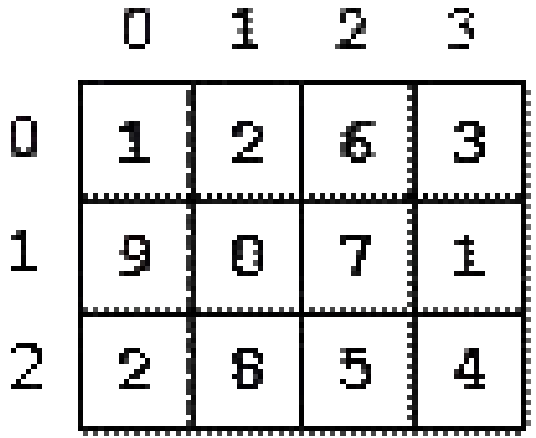
```
int columns =  matrix[0].length; // 4
```



Обхождане на матрица

Обхождаме матриците по същия начин, както и едномерните масиви, само че тук трябва да използваме вложени цикли (съответно за да минем през всеки ред и всяка колона):

```
for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){

                matrix[i][j] += 1;

        }
}
```



Задача

Имате предварително въведени стойности от цели числа, принадлежащи на интервала [10..99]. числата са въведени в матрица с размери 6 реда и 6 колони.
Да се състави програма, чрез която се намира произведението на елементи в ред с четни номера: 2, 4 и 6 по отделно.

Матрица:
1,15,13,85,63,16,
21,1,5,41,25,98,
46,32,71,55,31,12,
45,2,46,81,11,63,
12,52,29,49,17,59,
53,1,6,64,18,93
Изход:
21,1,5,41,25,98 произведението от елементите на колоната е: (произведението от числата в реда)
45,2,46,81,11,63 произведението от елементите на колоната е: (произведението от числата в реда)
53,1,6,64,18,93 произведението от елементите на колоната е: (произведението от числата в реда)

Методи

Какво е метод ?

Методите са, най-просто казано, именувана последователност от действия. Те ни позволяват да разделим програмата на подпрограми, всяка от които решава отделен проблем/задача..

Защо са важни методите ?

- за да не повтаряме код
- за да бъде кодът ни по-ясен и по-добре структуриран
- за да бъде лесно достъпен, без да се налага да пишем много

Какво е деклариране на метод ?

Деклариране на метод – създаване на метод. Методите се създават само **в рамките на някакъв клас и извън други методи.**

Деклариране

Декларацията на метод се състои в следните три задължителни неща:
<тип> <име> (<параметър1>,<параметър2>…<параметърN>),
където:

- <тип> е типът на променливата, която ни връща като резултат методът, независимо дали примитивен (int, long, double, etc.) или референтен (масиви, стрингове и др.)
- <име> е името на метода
- <параметър1>...<параметърN> са променливите, върху които трябва да извършим определени действия, за да решим задачата

Засега в декларацията на метода ще слагаме и public static, като по-натам ще видим какво правят те.

Конвенция за именоване на методи

Имената на методите трябва да започват с малка буква и да бъдат с т.нар. CamelCase - всяка следваща дума в името на метода започва с главна буква, а също така и добре да описват какво прави методът ни.

Имплементиране на метод

След като сме декларирали нашия метод, е време да опишем какво искаме да прави. Това се случва в т.нар. “тяло” на метода или пространството между къдравите скоби:

```
class Example {
    public static void max(int a, int b){
        int max = 0;
        if (a > b)
            max = a;
        else
            max = b;
        System.out.println(max);
    }
}
```

Извикване

Да извикаме един метод, означава да “задействаме” кода в него с определени стойности, за да решим дадена задача. Извикването се извършва, като напишем името на метода, последвано от скоби и списък с точни стойности на параметрите (ако има такива), за които искаме да решим задачата. Извикването се извършва само в тялото на някакъв клас. Може да се извикват методи и в рамките на други методи.

Параметри vs. аргументи

Когато създаваме един метод, посочваме списък с параметри - данните, необходими за решаването на задачата. Когато извикваме един метод, посочваме списък с аргументи - конкретните стойности, за които искаме да решим задачата.

```
class Example {
    public static void sum(int a, int b) {          // a, b - параметри
        System.out.println(a + b);
    }
    public static void main(String[] args){
        sum(4, 5);                  // 4, 5 - аргументи
    }
}
```

Връщане на стойност

Връщането на стойност става чрез **return**. Върнатата стойност трябва да е от същия тип като типа, посочен в декларацията:

```
class Example {
    public static int sum(int a, int b){
        int result = a + b;
        return result;
    }
}
```

Присвояване на стойност

Когато методът ни връща стойност, тази стойност трябва да бъде присвоена, иначе компилаторът ни хвърля грешка. По тази си особеност, извикването на методи много наподобява създаването на изрази:

```
class Example {
    public static int sum(int a, int b){
       return a + b;
    }
    public static void main(String[] args){
        int result1 = sum(4, 5);
        int result2 = 4 + 5;
    }
}
```

Задача

Напишете програма, която решава уравнението:

ax + b =0

- Ако числото **a ≠ 0** , то уравнението ax + b = 0 , има единствено решение и то е : x = -b/a

- Ако **a = 0 и b=0** , то уравнението 0.x + 0 = 0 , има безбройно много решения

,защото всяко произволно число x го превръща във вярно числово равенство .

- Ако **a = 0** ,а **b ≠ 0** ,то уравнението 0.x + b = 0 няма решение

Символни низове

Символни низове в Java

Символните низове (strings, стрингове) са поредица от символи.

В Java за обработка на низове се използва класът String.

String course = “Intro to Java”;

Символните низове се записват като последователност от символи, оградена в кавички.

Кавичките **не** са част от стойността на низа!

Особености

String е клас, а не прост тип (каквито например са int или boolean).

- Променливите могат да имат стойност null.

- Сравняват се по различен начин от простите типове.

- Класът String има методи за различни действия с низове (разгледайте ги).

Някои методи

- Можем да достъпваме символите на низа по индекси. Индексирането започва от 0, както при масивите:

```
String country = “Bulgaria”;

System.out.println(country.charAt(2)); // l
```

- Можем да вземем дължината на низа:

```
System.out.println(country.length()); // 8
```

Пример

Прочетете името на потребителя от конзолата. Отпечатайте в конзолата първата и последната буква и дължината му.

За четене на символен низ от конзолата може да се използва методът readLine() на класа Scanner:

```
Scanner input = new Scanner(System.in);

String s = input.nextLine();
```

Извод

Винаги, когато работите с променливи от тип String, имайте предвид, че може да имат стойност null и винаги правете проверки!

Задаване на стойност

Задаване на стойност

Можем да присвоим стойност на променлива от тип String по няколко начина:

String name = “Ivan”;

String job = new String(“programmer”);

String two = 2 + “”;

Можем да преобразуваме друг тип в низ, като го съберем с низ.

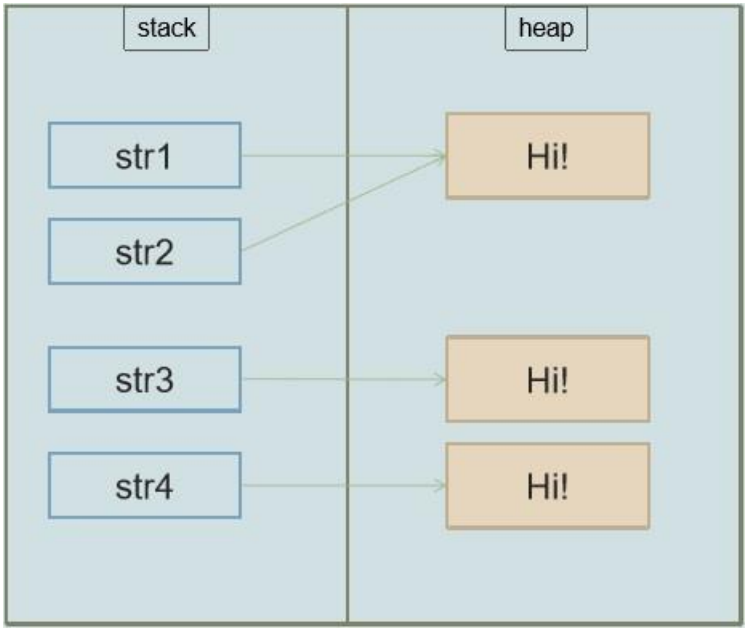
Сравняване на стрингове

Тъй като String е клас, символните низове в Java не се сравняват с оператора „==“.
При използване на „==“, се сравняват техните адреси, а не стойности.

Примери за сравнение

```
String str1 = “Hi!”;

String str2 = “Hi!”;

boolean equal =  str1 == str2;

System.out.println(equal); // true

String str3 = new String(“Hi!”);

String str4 = new String(“Hi!”);

equal =  str3 == str4; S

System.out.println(equal); // false
```

Представяне в паметта



Променливата не запазва в себе си стойността на низа, а сочи към място в паметта, където се пази тази стойност.

Сравнението с „==“ сравнява адресите, а не реалните стойности.

Правилно сравняване на низове

За сравнение на стойностите на низове се използва методът equals()

```
String str3 = new String(“Hi!”);
String str4 = new String(“Hi!”);
equal =  str3.equals(str4);
System.out.println(equal); // true
```

Слепване (конкатенация) на низове

Слепване (конкатенация) на низове

Пример:

String firstName = "Ivan";

String lastName = "Petrov";

Използване на знак „+“ за слепване на низове:

String fullName = firstName + " " + lastName; // Ivan Petrov

Използване на метода „concat()“ за слепване на низове:

String fullName2 = lastName.concat(", ").concat(firstName); // Petrov, Ivan

Важно

При класа String поредицата от символи , записана в паметта, не се изменя (нарича се immutable). При промяна на променливата, съдържанието не се променя, а се създава ново място в паметта, в което е записана новата стойност.

Затова не се препоръчва долепяне на низове в цикъл!

StringBuilder

StringBuilder

За работа в низове, когато имаме изменение на стойността на низа, се препоръчва използването на класа StringBuilder.

Пример

1. Създаваме нова променлива от клас StringBuilder:

```
StringBuilder builder = new StringBuilder();
for (int i = 1; i < n; i++) {
    if (builder.length() > 0) {
        builder.append(", ");
    }
```

2. Използваме метод append(), за да добавяме в края на низа:

```
     builder.append(i);
}
```

3. Използваме метод toString(), за да преобразуваме резултата в String:

```
return builder.toString();
```

Форматиране на низове

Метод String.format() :

Задава се шаблон с места, в които да се попълват данните от различни типове.

Задават се като следващи параметри на метода данните в последователността, в която трябва да бъдат попълнени.

String.format(“Name: %s, Age: %d years”, “Ivan”, 25);

//Name: Ivan, Age: 25 years

Задача

Прочетете текст от конзолата. Заменете всички главни букви с малки.

Пребройте колко пъти се съдържа буквата „а“ в текста. Заменете я с „А“.