Лабораторна робота №1
Введення в мову програмування Java.
Віртуальна машина Java. Базові засоби мови Java.

Мета лабораторної роботи:

1. Ознайомитись з роботою віртуальної машини Java.
2. Ознайомитись з JDK.
3. Навчитися створювати, компілювати та виконувати програми, написані на Java.
4. Ознайомитись з базовими типами та можливостями мови Java

Зміст роботи.

1. Теоретичні відомості

1.1 Java машина та JDK. Компіляція програми. [читати]

1.2 Примітивні типи даних у мові Java

1.3 Управління потоком виконання програми

1.4 Робота з рядками

1.5 Введення-виведення інформації з консолі]

2. Завдання на лабораторну роботу(увага - робота містить ДВА завдання)

Завдання 1

Для x що змінюється від a до b з кроком h, де h=(b-a)/k, k – задається користувачем (див. умову завдання 1), обчислити функцію f(x), використовуючи її розкладання в статечний ряд у трьох випадках:

1) для «точного» значення (за аналітичною формулою використовуючи стандартну бібліотеку java).
2) для заданого n (див. умову завдання 1),
3) для заданої точності e (запитати у користувача, не вище 1e-8);

Для порівняння знайти відносну похибку обчислення значення функції
o_погр = ABS((точ_знач – наближ_знач)/ точ_знач)
точ_знач – вважати значення розраховане першим способом

Результати розрахунків надрукувати в наступному вигляді:

```
       Обчислення функції "написати який"

     X      Y       Y1      Y2      погр1    погр2
   ...... ....    .....   .....     ....     ...
   ...... ....    .....   .....     ....     ...

   ...... ....    .....   .....     ....     ...
   ...... ....    .....   .....     ....     ...

Тут X значення параметра;
Y1 значення суми для заданого n;
Y2 значення суми для заданої точності;
Y-«точне» значення функції;
погр1, погр2 - відносні похибки наближених обчислень.
```

Значення a, b і f(x) згідно з варіантом можна подивитися тут

Математичне опис розкладання функції у ряд описано тут(Рекомендації щодо вибору способу розрахунку).

Приклад та додаткову інформацію можна переглянути тут

Завдання 2

Вказано рядок, що складається із символів. Символи поєднуються в слова. Слова один від одного відокремлюються одним або декількома пробілами або іншими роздільними символами. Виконати введення рядка з консолі та обробку її відповідно до свого варіанта.

Завдання згідно з варіантом можна подивитися тут

Приклади можна переглянути тут (Розбиття на слова)

роботу з рядками можна подивитися тут

3. Звіт

1. Постановка задач.
2. Варіант задання.
3. Математична модель (формули, якими виконуються обчислення доданків ряду).
4. Програма.5. Отримані результати роботи написаної програми.

Успіхів!

Значення a, b і f(x) згідно з варіантом

№	Y	a b	n	Y1 Y2
1	$y = 3^X$	$0{,}1 \le x \le 1$	10	$S = 1 + \frac{\ln 3}{1!}x + \frac{\ln^2 3}{2!}x^2 + \ldots + \frac{\ln^n 3}{n!}x^n$
2	$y = -\ln\left\|2\sin\frac{x}{2}\right\|$	$\frac{\pi}{5} \le x \le \frac{9\pi}{5}$	40	$S = \cos x + \frac{\cos 2x}{2} + \ldots + \frac{\cos nx}{n}$
3	$y = \sin X$	$0{,}1 \le x \le 1$	10	$S = x - \frac{x^3}{3!} + \ldots + (-1)^n \frac{x^{2n+1}}{(2n+1)!}$
4	$y = X arctgX - \ln\sqrt{1+x^2}$	$0{,}1 \le x \le 0{,}8$	10	$S = \frac{x^2}{2} - \frac{x^4}{12} + \ldots + (-1)^{n+1}\frac{x^{2n}}{2n(2n-1)}$
5	$y = e^x$	$1 \le x \le 2$	15	$S = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!}$
6	$y = e^{x\cos\pi/4} \cdot \cos(x\sin\pi/4)$	$0{,}1 \le x \le 1$	25	$S = 1 + \frac{\cos\frac{\pi}{4}}{1!}x + \ldots + \frac{\cos n\frac{\pi}{4}}{n!}x^n$
7	$y = \cos x$	$0{,}1 \le x \le 1$	10	$S = 1 - \frac{x^2}{2!} + \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$
8	$y = \frac{x\sin\pi/4}{1 - 2x\cos\pi/4 + x^2}$	$0{,}1 \le x \le 0{,}8$	40	$S = x\sin\frac{\pi}{4} + x^2\sin 2\frac{\pi}{4} + \ldots + x^n \sin n\frac{\pi}{4}$
9	$y = \frac{1}{4}\ln\frac{1+x}{1-x} + \frac{1}{2}arctgX$	$0{,}1 \le x \le 0{,}8$	3	$S = x + \frac{x^5}{5} + \ldots + \frac{x^{4n+1}}{4n+1}$
10	$y = e^{\cos x}\cos(\sin x)$	$0{,}1 \le x \le 1$	20	$S = 1 + \frac{\cos x}{1!} + \ldots + \frac{\cos nx}{n!}$
11	$y = (1+2x^2)e^{x^2}$	$0{,}1 \le x \le 1$	10	$S = 1 + 3x^2 + \ldots + \frac{2n+1}{n!}x^{2n}$
12	$y = -\frac{1}{2}\ln(1 - 2x\cos\frac{\pi}{3} + x^2)$	$0{,}1 \le x \le 0{,}8$	35	$S = \frac{x\cos\frac{\pi}{3}}{1} + \frac{x^2\cos 2\frac{\pi}{3}}{2} + \ldots + \frac{x^n\cos n\frac{\pi}{3}}{n}$
13	$y = \frac{1}{2}\ln x$	$0{,}2 \le x \le 1$	10	$S = \frac{x-1}{x+1} + \frac{1}{3}(\frac{x-1}{x+1})^3 + \ldots + \frac{1}{2n+1}(\frac{x-1}{x+1})^{2n+1}$
14	$y = \frac{1}{4}(x^2 - \frac{\pi^2}{3})$	$\frac{\pi}{5} \le x \le \pi$	20	$S = -\cos x + \frac{\cos 2x}{2^2} + \ldots + (-1)^n \frac{\cos nx}{n^2}$
15	$y = \frac{1+x^2}{2}arctgX - \frac{x}{2}$	$0{,}1 \le x \le 1$	30	$S = \frac{x^3}{3} - \frac{x^5}{15} + \ldots + (-1)^{n+1}\frac{x^{2n+1}}{4n^2-1}$
16	$y = \frac{\pi^2}{8} - \frac{\pi}{4}\|x\|$	$\frac{\pi}{5} \le x \le \pi$	40	$S = \cos x + \frac{\cos 3x}{3^2} + \ldots + \frac{\cos(2n-1)x}{(2n-1)^2}$

17	$y=\frac{e^x+e^{-x}}{2}$	$0{,}1 \le x \le 1$	10	$S=1+\frac{x^2}{2!}+\ldots+\frac{x^{2n}}{(2n)!}$
18	$y=\frac{1}{2}-\frac{\pi}{4}\lvert\sin x\rvert$	$0{,}1 \le x \le 0{,}8$	50	$S=\frac{\cos 2x}{3}+\frac{\cos 4x}{15}+\ldots+\frac{\cos 2nx}{4n^2-1}$
19	$y=e^{2x}$	$0{,}1 \le x \le 1$	20	$S=1+\frac{2x}{1!}+\ldots.+\frac{(2x)^n}{n!}$
20	$y=(\frac{x^2}{4}+\frac{x}{2}+1)e^{x/2}$	$0{,}1 \le x \le 1$	30	$S=1+2\frac{x}{2}+\ldots+\frac{n^2+1}{n!}(\frac{x}{2})^n$
21	$y=arctgX$	$0{,}1 \le x \le 1$	40	$S=x-\frac{x^3}{3}+\ldots.(-1)^n\frac{x^{2n+1}}{2n+1}$
22	$y=(1-\frac{x^2}{2})\cos x-\frac{x}{2}\sin x$	$0{,}1 \le x \le 1$	35	$S=1-\frac{3}{2}x^2+\ldots..+(-1)^n\frac{2n^2+1}{(2n)!}x^{2n}$
23	$y=2(\cos^2 x-1)$	$0{,}1 \le x \le 1$	15	$S=-\frac{(2x)^2}{2}+\frac{(2x)^4}{24}+\ldots+(-1)^n\frac{(2x)^{2n}}{(2n)!}$
24	$y=\ln(\frac{1}{2+2x+x^2})$	$-2 \le x \le -0{,}1$	40	$S=-(1+x)^2+\frac{(1+x)^4}{2}+\ldots+(-1)^n\frac{(1+x)^{2n}}{n}$
25	$y=\frac{e^x-e^{-x}}{2}$	$0{,}1 \le x \le 1$	20	$S=x+\frac{x^3}{3!}+\ldots+\frac{x^{2n+1}}{(2n+1)!}$

Розкладання функції до ряду

Дійсна функція £(x) називається аналітичною в точці e, якщо в деякій околиці $|x-\varepsilon|<R$ цієї точки функція розкладається в статечний ряд (ряд Тейлора):

$$f(x)=f(\varepsilon)+f'(\varepsilon)(x-\varepsilon)+\frac{f''(\varepsilon)}{2!}(x-\varepsilon)^2+\ldots+\frac{f^{(n)}(\varepsilon)}{n!}(x-\varepsilon)^n+\ldots \qquad (1)$$

При □=0 отримуємо ряд Маклорена:

$$f(x)=f(0)+f'(0)(x)+\frac{f''(0)}{2!}(x)^2+\ldots+\frac{f^{(n)}(0)}{n!}(x)^n+\ldots \qquad (2)$$

Різниця:

$$R_n(x)=f(x)-\sum_{k=0}^{n}\frac{f^{(k)}(\varepsilon)}{k!}(x-\varepsilon)^k$$

називається залишковим членом і є помилкою при заміні функції f(x) поліномом Тейлора.

Для ряду Маклорена

$$R_n(x)=\frac{f^{(n+1)}(\theta\cdot x)}{(n+1)!}x^{n+1} \quad \text{где } 0<\theta<1. \qquad (4)$$

Таким чином, обчислення значення функції можна звести до обчислення суми числового ряду

$$a1+a2+\ldots+an+\ldots \quad . \qquad (5)$$

Відомо, що числовий ряд називається схожим, якщо існує межа послідовності його приватних сум:

$$S = \lim_{\nu \to \infty} Sn, \qquad (6)$$

де $Sn = a1 + a2 +\ldots+an+\ldots$.

Число S називається сумою ряду. Вочевидь, що S=Sn+Rn ,

де Rn - залишок ряду, причому R>0 при n>1.

Для знаходження суми S ряду, що сходить (5) із заданою точністю eps потрібно вибрати число доданків n настільки великим, щоб мала місце нерівність |Rn|<eps.

Тоді приватна сума Sn приблизно може бути прийнята за точну суму ряду S (5).

Приблизно n вибрати так, щоб була нерівність $|S_{n+1}-S_n|<eps$ або $|a_n|<eps$.

Завдання зводиться до заміни функції статечним рядом та знаходження суми деякої кількості доданків S(x) = SUM(an (x,n)) при різних параметрах підсумовування x . Кожна складова суми залежить від параметра x і номера n, що визначає місце цього доданку в сумі.

Зазвичай формула загального члена суми належить одному з наступних трьох типів:

а) $\frac{x^n}{n!}$; $(-1)^n \frac{x^{2n+1}}{(2n+1)!}$; $\frac{x^{2n}}{(2n)!}$;

б) $\frac{\cos(nx)}{n}$; $\frac{\sin(2n-1)x}{2n-1}$; $\frac{\cos(2nx)}{4n^2-1}$;

в) $\frac{x^{4n+1}}{4n+1}$; $(-1)^n \frac{\cos(nx)}{n^2}$; $\frac{n^2+1}{n!}(\frac{x}{2})^n$.

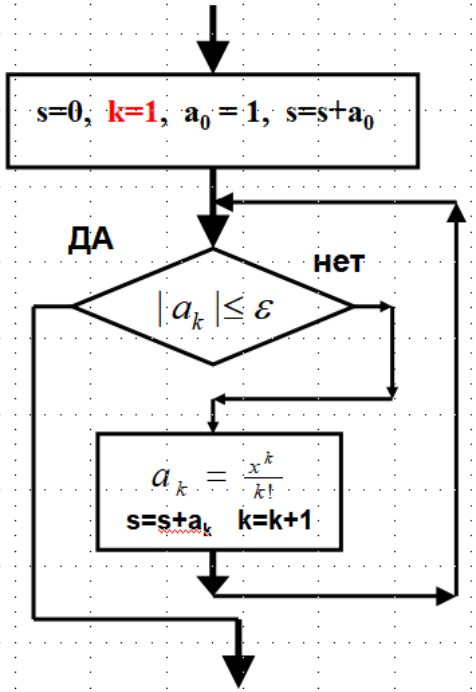
У разі а) для обчислення члена суми an доцільно використовувати рекурентні співвідношення, тобто виражати наступний член суми через попередній: $a_{n+1}=F(x, n)*a_n$. Це дозволить суттєво скоротити обсяг обчислювальної роботи. Крім того, обчислення члена суми за загальною формулою у ряді випадків неможливе (наприклад, через наявність n!).

У разі б) застосування рекурентних співвідношень недоцільне. Обчислення будуть найефективнішими, якщо кожен член суми обчислювати за загальною формулою $a_n=f(x, n)$.

У разі в) член суми доцільно подати у вигляді двох співмножників, один з яких обчислюється за рекурентним співвідношенням, а інший безпосередньо $a_n=F(x, n)*f_n(x,n)$, де $f_n=f_{n-1}(x,n)$.

Приклад – як «вважати» елементарні функції?

$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$	$x \in R$
$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots + (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!} + \ldots$	$x \in R$
$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!} + \ldots$	$x \in R$
$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots + (-1)^{n-1} \frac{x^n}{n} + \ldots$	$x \in (-1,1]$
$(1+x)^m = 1 + mx + \frac{m(m-1)}{1 \cdot 2} x^2 + \frac{m(m-1)(m-2)}{1 \cdot 2 \cdot 3} x^3 + \ldots$	$x \in [-1,1]$, если $m \geq 0$; $x \in (-1,1]$, если $-1 < m < 0$; $x \in (-1,1)$, если $m \leq -1$
$\operatorname{arctg} x = x - \frac{x^3}{3} + \frac{x^5}{5} - \ldots + (-1)^{n-1} \frac{x^{2n-1}}{2n-1} + \ldots$	$x \in [-1,1]$
$\frac{1}{1+x} = 1 - x + x^2 - x^3 + \ldots + (-1)^n x^n + \ldots$	$x \in (-1,1)$



$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$$

$$e^x = \sum_{k=0}^{\infty} a_k$$

$$a_k = \frac{x^k}{k!}$$

$$|a_k| \le \varepsilon$$

$$a_{n+1} = a_n + d \qquad n \in N$$

$$b_{n+1} = b_n \bullet q \qquad n \in N$$

n!=1*2*3*…….(n-2)*(n-1)*n = (n-1)! * n

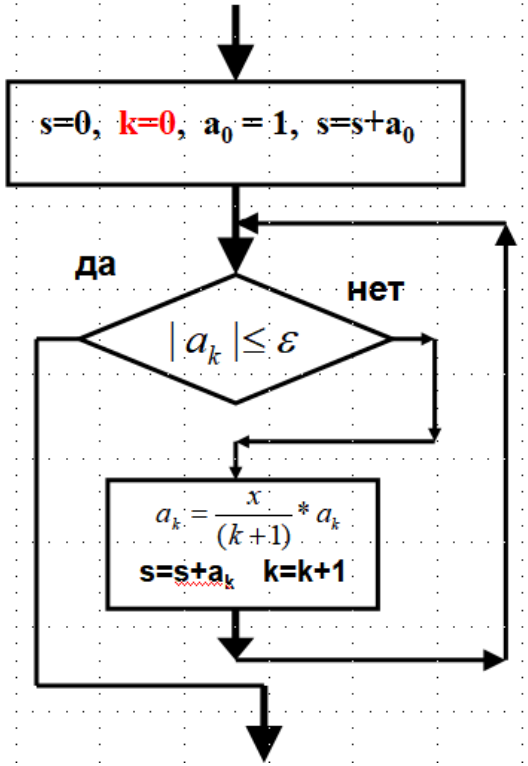
(n+1)! = (n+1)* n!

$X^{n+1} = X^n * X$

$$\frac{a_{k+1}}{a_k} = \frac{x^{k+1} * k!}{(k+1)! * x^k} = \frac{x^k * x * k!}{(k+1) * k! * x^k}$$

$$\frac{a_{k+1}}{a_k} = \frac{x}{(k+1)}$$

$$a_{k+1} = \frac{x}{(k+1)} *$$



```
/*
Приклад 1-ї лабораторної роботи.
       y(x)=exp(x)
*/
import java.math.*;
class lr1 {
// code application logic here
    public static void main(String [] args) {
        float a, b, x, er1, er2, y1, y2, y3, eps;
        int k_dot; // кількість рядків у таблиці
        int nsum;
        // вихідні дані
        // TODO організувати введення даних у діалозі
        a=1.0e0f; b = 2.0e0f; eps = 1e-5f;
        k_dot=10; nsum = 15;
        // Tab - клас друку таблиці
        // Digit - клас розрахунків Y(x) у різний спосіб
        Tab pr=new Tab("X", "Y1", "Y2", "Y3", "er1", "er2");
        //pr.print(.......)
        float h; // крок по X
        h = (ba) / (float) k_dot;
        // клас "розрахунок" методи ra1(x) ra2(x) ra3(x)
        Digit dig = New Digit (nsum, eps);
        for(x=a; x<=b; x=x+h){
            y1=dig.ra1(x);
            y2=dig.ra2(x);
            y3=dig.ra3(x);
            er1=Math.abs( (y1-y2)/y1 )*100;
            er2=Math.abs( (y1-y3)/y1 )*100;
            pr.print(x, y1, y2, y3, er1, er2);
        };
        pr.close();
    };
```

```
};

class Tab {
    private String n1, n2, n3, n4, n5, n6;
    private
       String br0="+------------";
       String br;
    public Tab(String z1, String z2,
               String z3, String z4,
               String z5, String z6) {
        n1=z1; n2=z2; n3=z3; n4=z4; n5=z5; n6=z6;
        br = "";
        for(int i=1; i <=6; i++)
          br = br + br0;
        br = br + "+";
        System.out.println(br);
       System.out.print("| " + n1 + " ");
       System.out.print("| " + n2 + " ");
        System.out.print("| " + n3 + " ");
        System.out.print("| " + n4 + " ");
        System.out.print("| " + n5 + " ");
        System.out.println("|"+n6+"|");
       System.out.println(br);
    };
    public void print(float p1, float p2,
                      float p3, float p4,
                       float p5, float p6) {
          System.out.printf("| %10.4e ", p1);
           System.out.printf("| %10.4e ", p2);
           System.out.printf("| %10.4e ", p3);
           System.out.printf("| %10.4e ", p4);
           System.out.printf("| %10.4e ", p5);
           System.out.printf("| %10.4e |\n", p6);
```

```
    };
    public void close() {
      System.out.println(br);

    }
};
class Digit {
   private float nsum, eps;
   public Digit(float nsum, float eps) {
       this.nsum=nsum;
       this.eps = eps;
   };
   public float ra1(float x){
      return (float) Math.exp ((double) x);
   };
   public float ra2(float x){
       float sum; int k; float a;
       sum=0.0f;
       a=1.0f;
       sum = sum + a;
       for(k=0; k <= nsum; k++) {
         a = a * x / (k+1);
         sum+=a;
       }
      return sum;
   };
   public float ra3(float x){
       float sum; int k; float a;
       sum=0.0f;
       a=1.0f;
       sum = sum + a;
       k=0;
       while ( Math.abs(a) > eps ) {
```

```
            a = a * x / (k+1);
            sum+=a;
            k++;
          }
        return sum;
    };
};
/*
Результат:
D:\java>java lr1
```

X	Y1	Y2	Y3	er1	er2
1,0000e+00	2,7183e+00	2,7183e+00	2,7183e+00	8,7709e-06	0,0000e+00
1,1000e+00	3,0042e+00	3,0042e+00	3,0042e+00	1,5873e-05	3,9681e-05
1,2000e+00	3,3201e+00	3,3201e+00	3,3201e+00	7,1810e-06	0,0000e+00
1,3000e+00	3,6693e+00	3,6693e+00	3,6693e+00	6,4977e-06	6,4977e-06
1,4000e+00	4,0552e+00	4,0552e+00	4,0552e+00	1,1759e-05	3,5276e-05
1,5000e+00	4,4817e+00	4,4817e+00	4,4817e+00	0,0000e+00	1,0640e-05
1,6000e+00	4,9530e+00	4,9530e+00	4,9530e+00	9,6272e-06	0,0000e+00
1,7000e+00	5,4739e+00	5,4739e+00	5,4739e+00	8,7110e-06	3,4844e-05
1,8000e+00	6,0496e+00	6,0496e+00	6,0496e+00	0,0000e+00	7,8821e-06
1,9000e+00	6,6859e+00	6,6859e+00	6,6859e+00	7,1320e-06	1,4264e-05

Завдання 2

1. Перевірити чи є рядок паліндромом. (Паліндром - це вираз, який читається однакова зліва направо та праворуч наліво. Пробіли ігноруються).

2. Надрукувати найдовше і найкоротше слово в рядку.

3. Надрукувати всі слова, які не містять голосних букв.

4. Надрукувати всі слова, які містять одну цифру.

5. Друкувати всі слова, які збігаються з її першим словом.

6. Перетворити рядок таким чином, щоб спочатку в ньому були надруковані лише літери, а потім тільки цифри, не змінюючи порядку символів у рядку.

7. Перетворити рядок так, щоб усі літери в ньому були відсортовані за зростанням.

8. Перетворити рядок так, щоб усі цифри в ньому були відсортовані за спаданням.

9. Перетворити рядок так, щоб усі слова в ньому стали ідентифікаторами, слова, що складаються лише з цифр, - видалити.

10. Надрукувати всі слова-паліндроми, які є в цьому рядку (див. 1 варіант).

11. Перетворити рядок таким чином, щоб на початку були записані слова, що містять тільки цифри, потім слова, що містять тільки літери, а потім слова, які містять і літери і цифри.

12. Перетворити рядок таким чином, щоб усі слова у ньому були надруковані навпаки.

13. Перетворити рядок таким чином, щоб літери кожного слова в ньому були відсортовані за зростанням.

14. Перетворити рядок таким чином, щоб цифри кожного слова в ньому були відсортовані за спаданням.

15. Перетворити рядок таким чином, щоб у ньому залишилися лише слова, що містять літери та цифри, решту слів видалити.

16. Визначити яке слово зустрічається у рядку найчастіше.

17. Визначити якісь слова зустрічаються в рядку по одному разу.

18. Усі слова рядка, які починаються з літери, відсортувати за абеткою.

19. Всі слова рядки, які починаються з цифри відсортувати за спаданням.

20. Видалити з рядка всі слова, які є ідентифікаторами.

21. Перевірити чи є введений рядок правильним текстовим поданням речового числа у науковій нотації.

22. Підрахувати частоти різних слів у рядку.

23. Здійснити шифрування та дешифрування рядка шляхом циклічного обертання кодів символів.

24. Перевірити, чи введений рядок є ідентифікатором мови програмування.

25. Зробити послівний переклад всіх слів рядка. Підстановковий словник може містити трохи більше 10 слів, можна враховувати зміна форм слова.

Приклад завдання 2 (розбивка на слова)

```
import java.io.*;
import java.util.*;
import java.util.Scanner;

/* string */
class words {
   private static BufferedReader in =
               New BufferedReader(New InputStreamReader(System.in));
   public static void main(String ar[]) {
      int i, k;
      String s= new String();
      String word=new String();
      while(true) {
         System.out.println("input string (empty - end):");
         try {
            s=in.readLine(); //Читаємо з клавіатури
         } catch (Exception e) {
            System.out.println( "*** Error of Inupt console .....");
            s="";
        };
        System.out.println("input string is:");
        System.out.println("<"+s+">");
        if ( s.length() == 0 ) break;
        k=1;
             /* Розбір на слова */
        StringTokenizer st = new StringTokenizer(s, "\t\n\r,.;");
        while(st.hasMoreTokens()) {
           // Отримуємо слово і щось робимо з ним, наприклад,
           // просто виводимо на екран
           word=st.nextToken();
           System.out.println(k+" \t <" + word + ">");
           k++;
        };
      };
      System.out.println("Test words ending");
   };
};
```

```
/**********приклад читання з використанням Scanner
Scanner in =new Scanner( System.in );
int i = 0;
do {
    try {
       System.out.println( "Введіть число: " );
       // in.nextInt(); in.nextLine();
        Double op1 = in.nextDouble();
        System.out.println( op1 * op1 );
     } catch (Exception e) {
         System.out.println( "Число введено неправильно!");
         in.nextLine();
    }
} while (i++ <= 10);
********************/
```

Коротко про рядки

У рядках Java представлені класом `String`. Вони є незмінними (immutable) об'єктами: будь-яка операція над рядком створює новий об'єкт, не змінюючи оригінал.
Основні операції та методи

- **Створення:**
  - Літерал:`String s = "Java";`(рекомендується використовувати String Pool).
  - Конструктор:`String s = new String("Java");`(Створює новий об'єкт у купі).
- **Порівняння:**
  - `equals(str)`- Перевірка на рівність вмісту (не використовуйте==для рядків!).
  - `equalsIgnoreCase(str)`- Порівняння без урахування регістру.
  - `compareTo(str)`- лексикографічне порівняння.
- **Пошук та перевірка:**

    - `length()` - Довжина рядка.
    - `contains(sub)` - Чи містить підрядок.
    - `indexOf("a")`/`lastIndexOf("a")` - Індекс першого / останнього входження.
    - `startsWith(prefix)`/`endsWith(suffix)` - Перевірка початку/кінця рядка.
- **Модифікація (створюють новий рядок):**
    - `substring(begin, end)` - Вилучення частини рядка.
    - `replace(old, new)` - Заміна символів або підрядків.
    - `доLowerCase()`/`toUpperCase()` - Зміна регістра.
    - `trim()`/`strip()` - Видалення пробілів по краях.
    - `split(regex)` - Розбиття рядка в масив по роздільнику.

Важливі класи-помічники

1. **`StringBuilder`**— використовуйте для частого склеювання рядків у циклах (працює швидше та заощаджує пам'ять, оскільки він змінюється).
2. **`String.format()`**— зручний для створення складних рядків за шаблоном (наприклад,`%s`для рядків,`%d`для чисел).

Рядки в Java: створення, методи та операції

Ось короткі приклади найчастіших операцій:
1. Об'єднання та форматування
java
```
String name = "Іван";
int age = 25;

// Форматування
String info = String.format("Ім'я: %s, Вік: %d", name, age);

// Ефективне складання в циклі
StringBuilder sb = новий StringBuilder();
for (int i = 0; i < 5; i++) {
    sb.append(i).append(" ");
}
String result = sb.toString(); // "0 1 2 3 4"
```

2. Порівняння та пошук
java
```
String s1 = "Java";
String s2 = "java";

System.out.println(s1.equals(s2)); // false
System.out.println(s1.equalsIgnoreCase(s2)); // true
```

```
String text = "Вчимо Java разом";
boolean hasJava = text.contains("Java"); // true
int index = text.indexOf("Java"); // 5
```

3. Поділ та вилучення

```
String data = "яблуко, банан, груша";
String[] fruits = data.split(","); // ["яблуко", "банан", "груша"]

String part = data.substring(0, 6); // "яблуко"
```

Використовуйте код із обережністю.

4. Очищення та заміна

```
String messy = "Зайві прогалини";
String clean = messy.trim(); // "Зайві прогалини"

String replaced = "Кіт сидить на вікні".replace("Кіт", "Пес");
```

Щоб не наступати на класичні «граблі» початківців, важливо розуміти різницю між вмістом та посиланням.

1. Чому==- це пастка

Оператор==порівнює адреси у пам'яті, а не літери у рядку.

- **String Pool:**Java зберігає літерали (рядки в лапках) у спеціальну область пам'яті. Якщо дві змінні створені як`"Java"`, можуть вказувати на той самий об'єкт.
- **New String:**Якщо рядок створено через`new`, вона завжди займає нову адресу.

java

```
String s1 = "Java";
String s2 = "Java";
String s3 = new String("Java");

System.out.println(s1 == s2); // true (посилаються однією об'єкт у пулі)
System.out.println(s1 == s3); // false (різні об'єкти у пам'яті)
System.out.println(s1.equals(s3)); // true (вміст однаковий)
```

Правило:Завжди використовуйте`.equals()`для порівняння рядків.

---

2. Регулярні вирази (Regex)
Якщо потрібно не просто знайти слово, а перевірити формат (наприклад, пошту чи телефон), на допомогу приходить метод`matches`або клас`Pattern`.
java
```
String email = " test@example.com ";
// Перевірка: чи є собачка та точка
boolean isValid = email.matches(".+@.+\\..+");

String text = "Ціна: 500 руб, Знижка: 100 руб";
// Замінити всі цифри на зірочки
String hidden = text.replaceAll("\d+", "***");
// "Ціна: *** руб, Знижка: *** руб"
```

3. Корисний лайфхак: String.join
Якщо у вас є масив або список, і їх потрібно склеїти через кому:
java
```
String[] tags = {"java", "programming", "backend"};
String joined = String.join("#", tags);
// "java #programming #backend"
```

Text Blocks(для зручності) та Regex-символи (для сили).
1. Текстові блоки (Java 15+)
Забудьте про склеювання рядків через`+`та купі символів`\n`якщо потрібно вставити шматок HTML, JSON або SQL-запит. Використовуйте потрійні лапки:
java
```
String json = """
    {
        "course": "Java",
        "status": "active"
    }
    """;
// Зберігає форматування та відступи автоматично
```

2. Короткий довідник Regular Expressions (Regex)
Ці символи вставляють у методи`matches()`,`split()`або`replaceAll()`:

- **Класи символів:**
  - `\d`- Будь-яка цифра (0-9).
  - `\\D`- Все, що не цифра.
  - `\w`- літера, цифра або_.
  - `\s`- Пробіл, табуляція або перенесення рядка.
  - `.`- Загалом будь-який символ.

- **Квантифікатори (кілька разів повторювати):**
  - `*`- 0 або більше разів.
  - `+`- 1 або більше разів.
  - `?`- 0 або 1 раз.
  - `{n, m}`- Від n до m разів.
- **Межі:**
  - `^`- Початок рядка.
  - `$`- Кінець рядка.

Приклад із життя:
java
```
String phone = "+7 (999) 123-45-67";
// Видаляємо все, крім цифр, щоб отримати чистий номер
String cleanPhone = phone.replaceAll("[^\\d]", ""); // "79991234567"
```

3. Метод`intern()`- "ручний" закид у пул
Якщо у вас є рядок, створений через `new String()`, але ви хочете, щоб вона переїхала в String Pool для економії пам'яті або порівняння через`==`, використовуйте `s.intern()`. Це поверне посилання на рядок із пулу.
як ефективно переводити рядки у числа (і навпаки)

Найнадійніший спосіб - використовувати статичні методи класів-оберток (`Integer`,`Double`і т.д.).

1. З рядка до числа
Для кожного типу є метод `parse...`.
Важливо: якщо рядок не числом, Java викине `NumberFormatException` тому краще використовувати блок `try-catch`.
java
```
String str = "123";

try {
    int num = Integer.parseInt(str); // У ціле
    double d = Double.parseDouble("12.5"); // У дрібне
} catch (NumberFormatException e) {
    System.out.println("Помилка: рядок не є числом!");
}
```

2. З числа в рядок
Тут є три популярні способи:

- **`String.valueOf(num)`** - Найшвидший і правильний спосіб.
- `Integer.toString(num)` - Працює аналогічно.
- `""+num` — найледачіший спосіб (не рекомендується для високонавантаженого коду, оскільки створює зайві об'єкти).

3. Java 8+ та Stream API (для масивів)
Якщо у вас є список рядків, які потрібно перетворити на список чисел:
java

```
List<String> strings = List.of("1", "2", "3");
List<Integer> numbers = strings.stream()
                               .map(Integer::valueOf)
                               . toList ();
```

---

1. Теоретичні відомості

1.1 Java машина та JDK. Компіляція програми. [читати]

1.1.1 Що таке Java та як вона працює?

Java - це одна з мов програмування, яка відноситься до об'єктно-орієнтованих мов. Він розроблений компанією Sun (зараз підрозділ Oracle) і є платформенно-незалежним. Це означає, що програма, написана на Java, однаково виконуватиметься і під Windows, і під Linux, і під іншими ОС. Досягається це так - текст програми перекладається з допомогою компілятора над рідні для процесора команди, а т.зв. байт-коди віртуальної java-машини (JVM) Коди віртуальної машини однакові на будь-якій платформі, і саме тому та сама програма і працюватиме на різних платформах. Сама ж віртуальна машина від платформи, звичайно, залежить – віртуальна машина для Windows відрізняється від віртуальної машини для інших ОС.

Методичні вказівки до виконання лабораторних робіт ми розглядатимемо стосовно ОС Windows. Але допускається використання будь-яких інших платформ.

Існує кілька основних видів програм на Java - консольні Java-програми, програми з графічним інтерфейсом користувача (GUI), аплети та ін. Деякі їх виконуються як самостійні програми, аплети виконуються у браузері, деякі програми (сервлети) виконуються з допомогою спеціальних додаткових програм - т.зв. серверів додатків.

Ми при вивченні базових можливостей об'єктно-орієнтованого програмування мовою Java будемо використовувати консольні програми, оскільки

вони є найпростішими, і при реалізації алгоритму дозволяють не відволікатися на необхідність реалізації інтерфейсу з користувачем.

Для виконання відкомпільованих програм, написаних на Java, необхідне середовище виконання JRE - Java Runtime Enviropment. Якщо ж нам необхідно здійснювати і розробку програм на мові Java, необхідно також наявність компілятора, бібліотек базових класів і деякі додаткові утиліти. Все це входить до пакету JDK - Java Development Kit. JDK містить навіть JRE.

1.1.2 Що таке JDK та як його встановити?

JDK (Java Developer Kit) – це набір програм та утиліт, призначений для програмування на Java. До нього входить ряд утиліт. Ось деякі з них:

- Компілятори javac. Саме він і переводить текст програми на Java у байт-коди віртуальної машини.
- Інтерпретатор java З його допомогою запускати відкомпільовані в байт-коди програми. Він містить у собі JVM (Віртуальну машину Java).
- Утиліта javadoc. Вона призначена для створення документації.

Є ще й інші утиліти.

JDK – набір утиліт для роботи з командного рядка. І не містить у собі IDE (Integrated Development Enviropment) – інтегроване середовище розробки. Тому готувати вихідні тексти програм доведеться за допомогою сторонніх текстових редакторів. Про підготовку вихідних кодів розказано у розділі.

Завантажити JDK можна із сайтуhttp://www.oracle.com/technetwork/java/javase/downloads/

Після завантаження необхідно інсталювати пакет на комп'ютері. За замовчуванням установка файлів проводиться, наприклад, у директорію "C:\Program Files\Java\jdk1.8.0_31\".В імені директорії цифра 8 позначає версію мови Java, а 31 - номер складання (build, у термінах Java - update). Тобто. у цьому випадку ми встановили JDK 8 Update 31.

Необхідно зауважити, що на відміну від JRE, JDK при встановленні автоматично шляху до виконуваних файлів (віртуальної машини, компілятора, засоби документування) не прописує.

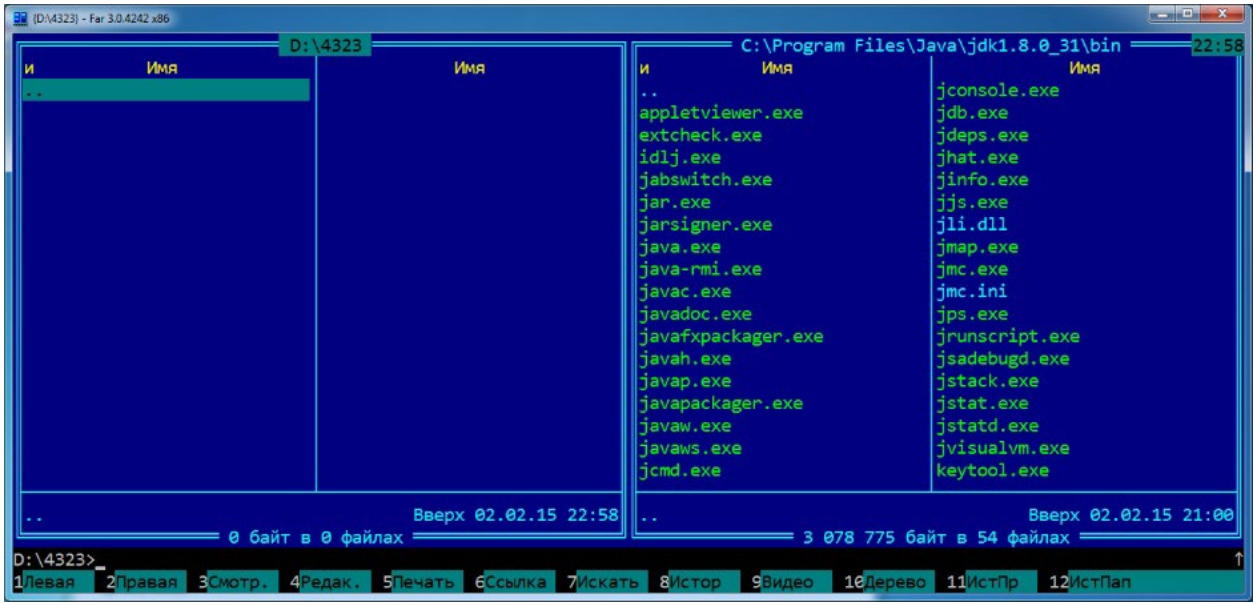
Для виконання "компіляції" вихідної програми використовується компілятор javac.exe, в нашому випадку розміщений в "C: Program Files Java Jdk1.8.0_31bin".

1.1.3 Підготовка Java програми до компіляції та виконання

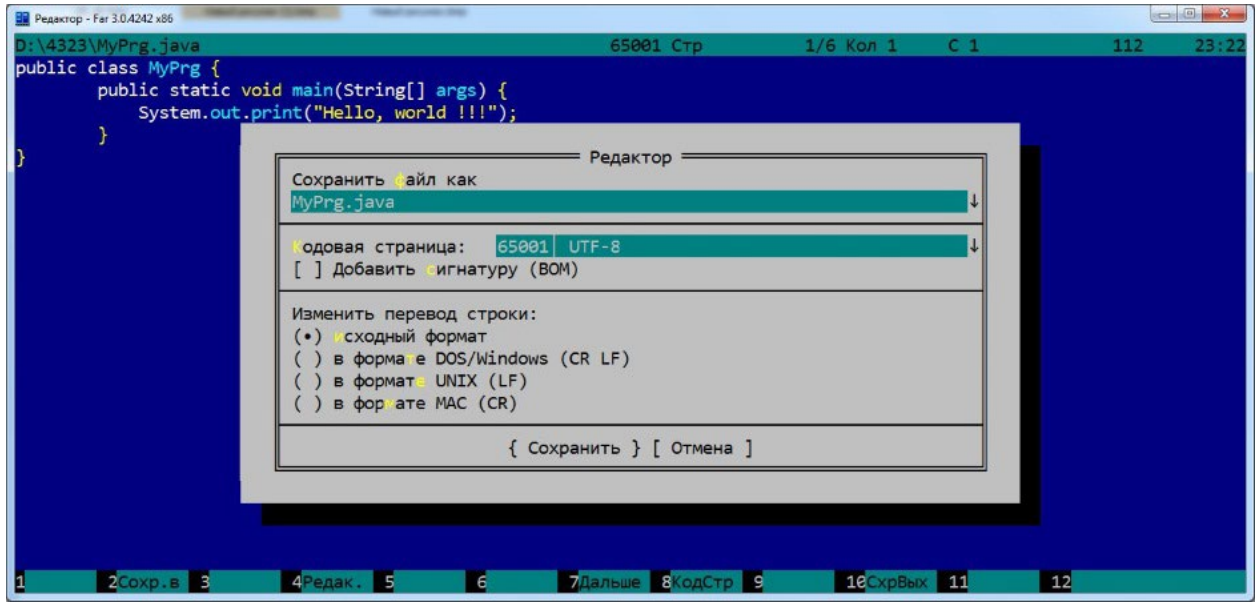
Процес створення програми Java досить простий. Спочатку створюється файл із вихідним текстом програми. Він має розширення java, і його рекомендовано створювати у кодуванні Unicode (utf-8). За допомогою компілятора javac файл з вихідним кодом компілюється у файл із байт-кодом (за відсутності помилок), який має розширення class. Потім, за допомогою віртуальної машини java файл із байт-кодом запускається на виконання.

Найбільш простий метод підготовки та виконання Java програми полягає в наступному:

*Крок 1*На робочому диску (наприклад, D) створіть свою інструментальну папку (наприклад, відповідно до номера академічної групи - 1229). Для простоти в консольному режимі рекомендується використовувати систему імен 8.3. Виконати ці дії можна або за допомогою команд консолі (CMD), або за допомогою файлового менеджера (наприклад,FAR) див. малюнок.



*Крок 2*Виберіть зручний для Вас консольний текстовий редактор. Це може бути Блокнот Windows та вбудований редактор файлового менеджера (наприклад, FAR), інші текстові редактори. Вихідні тексти, написані на Java рекомендується готувати в кодуванні Unicode (зокрема, UTF-8), причому, без (!!!) мітки порядку байтів (byte order mark – BOM). Найчастіше використовувані Блокнот Windows і вбудований редактор файлового менеджера (наприклад, FAR) не завжди коректно працюють з такими файлами. У зв'язку з цим, якщо виникають труднощі з підготовкою вихідного коду, рекомендується скористатися безкоштовним текстовим редактором з відкритим кодом Notepad ++. Завантажувати рекомендується не інсталятор, а архів (наприклад 7zip), оскільки після розархівування вийде портативна версія. Завантажити можна зцьогосайту у розділідownloads. У меню Опції - Налаштування можна на вкадці "Загальні" змінити інтерфейс програми (на російську, українську - за замовчуванням - англійську). Крім того, у текстовому редакторі менеджера FAR, командою SaveAs (F2), можна вибрати кодування файлу та інші праметри - див. малюнок.



*Крок 3*Оскільки будь-яку більш-менш складну програму доведеться налагоджувати в процесі написання, то операції компіляції та виклику на виконання викликатимуться досить часто. Для зручності виконання цих операцій підготуємо два командні файли, які спростять виконання зазначених команд. Для компіляції файлу з вихідним текстом створимо командний файл jc.bat такого змісту:

"C:\Program Files\Java\jdk1.8.0_31\bin\javac.exe" %1.java

Для виконання файлу з байт-кодом (файл з розширенням class) створимо командний файл jr.bat такого змісту:

"C:\Program Files\Java\jdk1.8.0_31\bin\java.exe" %1

Тепер, якщо ми маємо файл з вихідним текстом програми MyPrg.java, то для його компіляції необхідно виконати команду:

jc.bat MyPrg

Якщо компіляція пройшла успішно, і було створено файл з байт-кодом MyPrg.class, то його виконання необхідно виконати команду:

jr.bat MyPrg

1.1.4 Перша програма Java

Перша програма, за давно вкоріненою традицією, буде HelloWorld. Нижче наводиться її текст, який треба підготувати згідно з рекомендаціями у розділі 1.1.3. Збережіть його у файлі MyPrg.java:

```
public class MyPrg {
        public static void main(String[] args) {
            System.out.print("Hello, world !!!");
        }
}
```

Увага! Ім'я файлу та ім'я класу з методом main повинні збігатися. При цьому імена Java чутливі до регістру літер (реєстрозалежні). Так що, наприклад, MyPrg та myprg – різні речі.

Декілька слів за текстом програми. По-перше, зверніть увагу на слово class. Будь-яка програма Java використовує класи. По-друге, обов'язково має бути метод (функція) main. Саме з нього все починається. У нашому випадку main має декілька модифікаторів. Модифікатор public означає, що цей метод буде видно зовні класу. Модифікатор static приблизно означає, що метод main можна викликати не для конкретного екземпляра класу, а на початку програми, коли ніякого екземпляра класу ще немає. І, нарешті, void означає, що метод main не повертає ніякого значення. У рядку

System.out.println("Hello World!");

викликається метод println, який і виводить на екран відповідний напис. Цей метод береться із простору імен System.out.

Після того, як текст набраний (нагадаємо, що ми зберігаємо його у файлі з ім'ям MyPrg.java), його потрібно відкомпілювати та виконати. Для цього виконаємо команди:

jc MyPrg
jr MyPrg

Прості типи в Java

Прості типи Java не є об'єктно-орієнтованими, вони аналогічні простим типам більшості традиційних мов програмування. У Java є вісім простих типів: - byte, short, int, long, char, float, double і boolean. Їх можна розділити на чотири групи:

1. Цілі. До них відносяться типи byte, short, int та long. Ці типи призначені для цілих чисел зі знаком.
2. Типи з плаваючою точкою — float та double. Вони служать для представлення чисел, що мають дрібну частину.
3. Символ тип char. Цей тип призначений для представлення елементів із таблиці символів, наприклад, літер або цифр.
4. Логічний тип boolean. Це спеціальний тип, використовуваний уявлення логічних величин.

У Java, на відміну від деяких інших мов, немає автоматичного приведення типів. Розбіжність типів призводить не до попередження при трансляції, а до повідомлення про помилку. Для кожного типу суворо визначено набори допустимих значень та дозволених операцій.

Цілі числа

У мові Java поняття беззнакових чисел відсутнє. Усі числові типи цієї мови знакові. Наприклад, якщо значення змінної типу byte дорівнює шістнадцятковому вигляді 0x80, це — число -1.

Відсутність Java беззнакових чисел вдвічі скорочує кількість цілих типів. У мові є 4 цілих типу, що займають 1, 2, 4 та 8 байтів у пам'яті. Для кожного типу - byte, short, int і long, є свої природні сфери застосування.

byte

Тип byte – це знаковий 8-бітовий тип. Його діапазон - від -128 до 127. Він найкраще підходить для зберігання довільного потоку байтів, що завантажується з мережі або файлу.

```
byte b;

byte з = 0x55;
```

Якщо не йдеться про маніпуляції з бітами, використання типу byte, як правило, слід уникати. Для нормальних цілих чисел, що використовуються як лічильники і в арифметичних виразах, набагато краще підходить тип int.

short

short – це знаковий 16-бітовий тип. Його діапазон - від -32768 до 32767. Це, ймовірно, найбільш рідко використовуваний Java тип, оскільки він визначений, як тип, в якому старший байт стоїть першим.

```
short s;

short t = 0x55aa;
```

int

Тип int служить уявлення 32-битных цілих чисел зі знаком. Діапазон допустимих для цього типу значень - від -2147483648 до 2147483647. Найчастіше цей тип даних використовується для зберігання звичайних цілих чисел зі значеннями, що сягають двох мільярдів. Цей тип чудово підходить для використання при обробці масивів та для лічильників. У найближчі роки цей тип буде чудово відповідати машинним словам не тільки 32-бітових процесорів, але й 64-бітових за допомогою швидкої конвеєризації для виконання 32-бітового коду в режимі сумісності. Щоразу, як у одному вираженні фігурують змінні типів byte, short, int і цілі літерали, тип всього висловлювання перед завершенням обчислень наводиться до int.

```
int i;

int j = 0x55aa0000;
```

long

Тип long призначений для представлення 64-бітових чисел зі знаком. Його діапазон допустимих значень досить великий навіть таких завдань, як підрахунок числа атомів у всесвіту.

```
long m;

long n = 0x55aa000055aa0000;
```

Не треба ототожнювати розрядність цілого чисельного типу з займаним ним кількістю пам'яті. Виконуючий код Java може використовувати для ваших змінних ту кількість пам'яті, яка вважатиме за потрібне, аби тільки їх поведінка відповідала поведінці типів, заданих вами. Фактично, нинішня реалізація Java з міркувань ефективності зберігає змінні типу byte і short як 32-бітових значень, оскільки цей розмір відповідає машинному слову більшості сучасних комп'ютерів (СМ – 8 біт, 8086 – 16 біт, 80386/486 – 32 біт.

Нижче наведено таблицю розрядностей та допустимих діапазонів для різних типів цілих чисел.

Ім'я	Розрядність	Діапазон
long	64	-9, 223, 372, 036, 854, 775, 808. 9, 223, 372, 036, 854, 775, 807
Int	32	-2, 147, 483, 648. 2, 147, 483, 647
Short	16	-32, 768. 32, 767
byte	8	-128.. 127

Числа з плаваючою точкою

Числа з плаваючою точкою, які часто називають іншими мовами речовими числами, використовуються при обчисленнях, в яких потрібне використання дробової частини. У Java реалізований стандартний (IEEE-754) набір типів для чисел з плаваючою точкою - float і double та операторів для роботи з ними. Характеристики цих типів наведені у таблиці.

Ім'я	Розрядність	Діапазон
double	64	1. 7e-308. 1. 7e+ 308
float	32	3. 4e-038.. 3. 4e+ 038

float

У змінних із звичайною, або одинарною точністю, що оголошуються за допомогою ключового слова float, для зберігання речового значення використовується 32 біти.

```
float f;

float f2 = 3. 14F; // зверніть увагу до F, т.к. за замовчуванням усі літерали double
```

double

У разі подвійної точності, яка задається за допомогою ключового слова double, для зберігання значень використовується 64 біти. Усі трансцендентні математичні функції, такі як sin, cos, sqrt, повертають результат типу double.

```
double d;

double pi = 3. 141592653589793238 46;
```

Приведення типу

Іноді виникають ситуації, коли у вас є величина певного типу, а вам потрібно її привласнити змінної іншого типу. Для деяких типів це можна зробити і без наведення типу, у таких випадках говорять про автоматичне перетворення типів. У Java автоматичне

перетворення можливе лише в тому випадку, коли точності представлення чисел змінної-приймача достатньо зберігання вихідного значення. Таке перетворення відбувається, наприклад, при занесенні літеральної константи або значення змінної типу byte або short змінну типу int. Це називається розширенням (widening) чи підвищенням (promotion), оскільки тип меншої розрядності розширюється (підвищується) до більшого сумісного типу. Розміру типу int завжди достатньо для зберігання чисел з діапазону, допустимого для типу byte, тому в подібних ситуаціях оператор явного приведення типу не вимагається. Зворотне у більшості випадків неправильне, тому для занесення значення типу int в змінну типу byte необхідно використовувати оператор приведення типу. Цю процедуру іноді називають звуженням (narrowing), оскільки ви явно повідомляє транслятору, що величину необхідно перетворити, щоб вона вмістилася в змінну потрібного вам типу. Для приведення величини до певного типу перед нею необхідно вказати цей тип, укладений у круглі дужки. У наведеному нижче фрагменті коду демонструється приведення типу джерела (змінної типу int) типу приймача (змінної типу byte). Якщо при такій операції ціле значення виходило за межі допустимого для типу byte діапазону, воно було б зменшено шляхом поділу по модулю на допустимий для byte діапазон (результат поділу по модулю на число - це залишок від поділу на це число).

```
int a = 100;

byte b = (byte) a;
```

Автоматичне перетворення типів у виразах

Коли ви обчислюєте значення виразу, точність, потрібна для зберігання проміжних результатів, часто повинна бути вищою, ніж потрібно для остаточного результату.

```
byte a = 40;

byte b = 50;

byte з = 100;

int d = a * b / c;
```

Результат проміжного виразу (a*b) цілком може вийти за діапазон допустимих для типу byte значень. Саме тому Java автоматично підвищує тип кожної частини виразу типу int, отже для проміжного результату (a* b) вистачає місця.

Автоматичне перетворення типу іноді може спричинити несподівані повідомлення транслятора про помилки. Наприклад, показаний нижче код, хоч і виглядає цілком коректним, призводить до повідомлення про помилку на фазі трансляції. У ньому ми намагаємося

записати значення 50*2, яке має чудово вміститися у тип byte, у байтову змінну. Але через автоматичне перетворення типу результату в int ми отримуємо повідомлення про помилку від транслятора - адже при занесенні int в byte може статися втрата точності.

```
byte b = 50;

b = b * 2;

^ Incompatible type for =. Explicit cast необхідний для конвертації int to byte.

(Несумісний тип =. Необхідно явне перетворення int в byte)

Виправлений текст:



byte b = 50;

b = (byte) (b * 2);
```

що призводить до занесення b правильного значення 100.

Якщо виразі використовуються змінні типів byte, short і int, те щоб уникнути переповнення тип всього виразу автоматично підвищується до int. Якщо ж у виразі тип хоча б однієї змінної - long, то й тип виразу теж підвищується до long. Не забувайте, що всі цілі літерали, наприкінці яких не стоїть символ L (або l), мають тип int.

Якщо вираз містить операнди типу float, то й тип виразу автоматично підвищується до float. Якщо ж хоча б один з операндів має тип double, то тип виразу підвищується до double. За замовчуванням Java розглядає всі літерали з плаваючою точкою, як такі, що мають тип double. Нижче наведена програма показує, як підвищується тип кожної величини у виразі для досягнення відповідності з другим операндом кожного бінарного оператора.

```
class Promote {

public static void main (String args []) { byte b = 42;

char з = 'a';
```

```
short s = 1024;

int i = 50 000;

float f = 5.67f;

double d=.1234;

double result = (f * b) + (i / c) - (d * s);

System. out. println((f*b)+"+"+(i/c)+"-"+(d*s));

System. out. println("result="+result);

}

}
```

Подвыражение f* b — це число типу float, помножене число типу byte. Тому його тип автоматично підвищується до float. Тип наступного виразу i/c (int, поділений на char) підвищується до int. Аналогічно цьому тип подвыражения d * s (double, помножений на short) підвищується до double. На наступному етапі обчислень ми маємо справу з трьома проміжними результатами типів float, int та double. Спочатку при додаванні перших двох тип int підвищується до float і виходить результат типу float. При відніманні з нього значення типу double тип результату підвищується до double. Остаточний результат всього виразу – значення типу double.

Символи

Оскільки в Java для представлення символів у рядках використовується кодування Unicode, розрядність типу char цією мовою - 16 біт. У ньому можна зберігати десятки тисяч символів міжнародного набору символів Unicode. Діапазон типу char - 0..65536. Unicode - це об'єднання десятків кодувань символів, він включає латинську, грецьку, арабську алфавіти, кирилицю і багато інших наборів символів.

```
char c;

char c2 = Oxf132;

char c3 = 'a';
```

```
char c4 = '\n';
```

Хоча величини типу char і не використовуються, як цілі числа, ви можете оперувати з ними так, якби вони були цілими. Це дає можливість скласти два символи разом, або інкрементувати значення символьної змінної. У наведеному нижче фрагменті коду ми, маючи базовий символ, додаємо до нього ціле число, щоб отримати символьне уявлення потрібної нам цифри.

```
int three = 3;

char one = '1';

char four = (char) (three + one);
```

В результаті виконання цього коду змінну four заноситься символьне уявлення потрібної нам цифри — '4'. Зверніть увагу - тип змінної one у наведеному вище виразі підвищується до типу int, так що перед занесенням результату до змінної four доводиться використовувати оператор явного приведення типу.

Тип boolean

У мові Java є простий тип boolean, що використовується для зберігання логічних значень. Змінні цього можуть набувати лише два значення — true (істина) і false (брехня). Значення типу boolean повертаються як результат всіма операторами порівняння, наприклад (a < b) - boolean - це тип, необхідний всіма умовними операторами управління - такими, як if, while, do.

```
boolean done = false;
```

У наведеному прикладі створюються змінні кожного з простих типів і виводяться значення цих змінних.

```
class SimpleTypes {

public static void main(String args []) {

byte b = 0x55;

short s = 0x55ff;

int i = 1000000;
```

```
long l = 0xffffffffL;

char з = 'a';

float f = .25f;

double d=.00001234;

boolean bool = true;

System.out.println("byte b = "+ b);

System.out.println("short s="+s);

System.out.println("int i = " + i);

System.out.println("long l = " + l);

System.out.println("char з =" + c);

System.out.println("float f = " + f);

System.out.println("double d="+d);

System.out.println("boolean bool="+bool);

} }
```

Запустивши цю програму, ви повинні отримати результат, наведений нижче:

```
З: \> java SimpleTypes

byte b = 85

short s = 22015
```

int i = 1000000

long l = 4294967295

char з = a

float f = 0.25

double d = 1.234e-005

boolean bool = true

Оператори

Оператори в мові Java - це спеціальні символи, які повідомляють транслятору про те, що ви хочете виконати операцію з деякими операндами. Деякі оператори вимагають одного операнда, їх називають унарними. Одні оператори ставляться перед операндами і називаються префіксними, інші — після, їх називають постфіксними операторами. Більшість операторів ставлять між двома операндами, такі оператори називаються інфіксними бінарними операторами. Існує тернарний оператор, який працює із трьома операндами.

У Java є 44 вбудовані оператори. Їх можна розбити на 4 класи - арифметичні, бітові, оператори порівняння та логічні.

Арифметичні оператори

Арифметичні оператори використовуються для обчислень так само, як у алгебрі (див. таблицю зі зведенням арифметичних операторів нижче). Допустимі операнди повинні мати числові типи. Наприклад, використовувати ці оператори для роботи з логічними типами не можна, а для роботи з типом char можна, оскільки Java тип char — це підмножина типу int.

Оператор	Результат	Оператор	Результат
+	Додавання	+ =	додавання з привласненням

-	віднімання (також унарний мінус)	-=	віднімання з присвоєнням
*	Розмноження	* =	множення із присвоєнням
/	Поділ	/=	розподіл із присвоєнням
%	розподіл за модулем	%=	розподіл за модулем з присвоєнням
++	Інкремент	-	декремент

Чотири арифметичні дії

Нижче як приклад наведена проста програма, що демонструє використання операторів. Зверніть увагу на те, що оператори працюють як з цілими літералами, так і зі змінними.

class BasicMath {public static void int a = 1 + 1;

int b = a * 3;

main(String args[]) {

int c = b/4;

int d = b – a;

int e = -d;

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

System.out.println("b = " + b);

System.out.println("c = " + c);

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

System.out.println("e=" + e);

} }
```

Виконавши цю програму, ви повинні отримати наведений нижче результат:

```
C: \> java BasicMath

a = 2

b = 6

c = 1

d = 4

e = -4
```

Оператор поділу за модулем

Оператор поділу по модулю або оператор mod позначається символом %. Цей оператор повертає залишок від поділу першого операнда на другий. На відміну від C++, функція mod Java працює не тільки з цілими, але і з речовими типами. Нижче наведена програма ілюструє роботу цього оператора.

```
class Modulus {

public static void main (String args []) {

int x = 42;

double y = 42.3;

System.out.println("x mod 10 = " + x % 10);
```

```
System.out.println("y mod 10 = "+ y % 10);

} }
```

Виконавши цю програму, ви отримаєте наступний результат:

```
3:\> Modulus

x mod 10 = 2

y mod 10 = 2.3
```

Арифметичні оператори присвоєння

Для кожного з арифметичних операторів є форма, в якій одночасно із заданою операцією виконується привласнення. Нижче наведено приклад, який ілюструє використання такого різновиду операторів.

```
class OpEquals {

public static void main(String args[]) {

int a = 1;

int b = 2;

int c = 3;

a + = 5;

b * = 4;

c + = a * b;

з % = 6;

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

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

System.out.println("c = " + c);

} }
```

А ось і результат, отриманий під час запуску цієї програми:

```
З:> Java OpEquals

a = 6

b = 8

з = 3
```

Інкремент та декремент

У С існує 2 оператори, званих операторами інкременту і декременту (++ і --) і є скороченим варіантом запису для складання чи віднімання з операнда одиниці. Ці оператори унікальні в тому плані, що можуть використовуватися як у префіксній, так і постфіксній формі. Наступний приклад ілюструє використання операторів інкременту та декременту.

```
class IncDec {

public static void main(String args[]) {

int a = 1;

int b = 2;

int c = ++b;

int d = a++;

c++;
```

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

System.out.println("b = " + b);

System.out.println("c = " + c);

System.out.println("d="+d);

} }
```

Результат виконання цієї програми буде таким:

```
C:\ java IncDec

a = 2

b = 3

c = 4

d = 1
```

Цілочисельні бітові оператори

Для числових типів даних — long, int, short, char і byte, визначено додатковий набір операторів, з допомогою яких можна перевіряти і модифікувати стан окремих бітів відповідних значень. У таблиці наведено зведення таких операторів. Оператори бітової арифметики працюють з кожним бітом як із самостійною величиною.

Оператор	Результат	Оператор	Результат
~	побітове унарне заперечення (NOT)		
&	побітове І (AND)	&=	побітове І (AND) із присвоєнням
|	побітове АБО (OR)	|=	побітове АБО (OR) із присвоєнням

^	побітове що виключає АБО (XOR)	^=	побітове що виключає АБО (XOR) з привласненням
>>	зрушення вправо	>> =	зрушення вправо із присвоєнням
>>>	зсув праворуч із заповненням нулями	>>>=	зрушення вправо із заповненням нулями із присвоєнням
<<	зрушення вліво	<<=	зрушення вліво з присвоєнням

Приклад програми, що маніпулює з бітами

У наведеній нижче таблиці показано, як кожен з операторів бітової арифметики впливає на можливі комбінації бітів своїх операндів. Наведений після таблиці приклад ілюструє використання цих операторів у програмі Java.

A	У	OR	AND	XOR	NOT A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

```
class Bitlogic {

public static void main(String args []) {

String binary[] = { "0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111", "1000", "1001", "1010", "1011", "1100"

"1110", "1111"};

int a = 3; // 0+2+1 або двійкове 0011
```

```
int b = 6; // 4+2+0 або двійкове 0110

int c = a | b;

int d = a & b;

int e = a^b;

int f = (~a & b) | (a & ~ b);

int g = ~a & 0x0f;

System.out.println(" a = " + binary[a]);

System.out.println(" b = " + binary[b]);

System.out.println(" ab = " + binary[c]);

System.out.println(" a&b = " + binary[d]);

System.out.println(" a^b = " + binary[e]);

System.out.println("~a&b|a^~Ь = " + binary[f]);

System.out.println(" ~a = " + binary[g]);

} }
```

Нижче наведено результат, отриманий під час виконання цієї програми:

```
Java BitLogic

a = 0011

b = 0110
```

a | b = 0111

a & b = 0010

a ^ b = 0101

~a & b | a & b = 0101

~a = 1100

Зрушення вліво та вправо

Оператор << виконує зрушення вліво всіх бітів свого лівого операнда число позицій, задане правим операндом. При цьому частина бітів у лівих розрядах виходить за межі та губиться, а відповідні праві позиції заповнюються нулями. У попередній главі вже йшлося про автоматичне підвищення типу всього виразу до int у тому випадку, якщо у виразі присутні операнди типу int або цілих типів меншого розміру. Якщо ж хоча б один з операндів у виразі має тип long, то й тип виразу підвищується до long.

Оператор >> означає в мові Java зсув праворуч. Він переміщає всі біти свого лівого операнда вправо число позицій, задане правим операндом. Коли біти лівого операнда висуваються за праву позицію слова, вони губляться. При зрушенні вправо звільняються старші (ліві) розряди числа, що зрушується, заповнюються попереднім вмістом знакового розряду. Таку поведінку називають розширенням знакового розряду.

У наступній програмі байтове значення перетворюється на рядок, що містить його шістнадцяткове уявлення. Зверніть увагу - зсунуте значення доводиться маскувати, тобто логічно множити на значення 0x0f, для того, щоб очистити біти, що заповнюються в результаті розширення знака, і знизити значення до меж, допустимих при індексуванні масиву шістнадцятирічних цифр.

```
class HexByte {

static public void main(String args[]) {

char hex[] = { '0', '1, '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f};

byte b = (byte) 0xf1;

System.out.println("b = 0x" + hex[(b >> 4) & 0x0f] + hex[b & 0x0f]);
```

```
} }
```

Нижче наведено результат роботи цієї програми:

```
3:\> java HexByte

b = 0xf1
```

Беззнакове зрушення вправо

Часто потрібно, щоб при зрушенні вправо розширення знакового розряду не відбувалося, а ліві розряди, що звільняються, просто заповнювалися б нулями.

```
class ByteUShift {

static public void main(String args[]) {

char hex[] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f'};

byte b = (byte) 0xf1;

byte c = (byte) (b >> 4);

byte d = (byte) (b >> 4);

byte e = (byte) ((b & 0xff) >> 4);

System.out.println("b = 0x" + hex(b >> 4) & 0x0f] + hex[b & 0x0f]);

System.out.println(“b >> 4 = 0x” + hex[(c >> 4) & 0x0f] + hex[c & 0x0f]);

System.out.println(“b >>> 4 = 0x” + hex[(d >> 4) & 0x0f] + hex[d & 0x0f]);

System.out.println(“(b & 0xff) >> 4 = 0x” + hex[(e >> 4) & 0x0f] + hex[e & 0x0f]);

} }
```

Для цього прикладу змінну b можна було б ініціалізувати довільним негативним числом, ми використовували число з шістнадцятковим уявленням 0xf1. Змінної з надається результат знакового зсуву b вправо на 4 розряди. Як і очікувалося, розширення знакового розряду призводить до того, що 0xf1 перетворюється на 0xff. Потім змінну d заноситься результат беззнакового зсуву b праворуч на 4 розряду. Можна було б очікувати, що в результаті d містить 0x0f, але насправді ми знову отримуємо 0xff. Це результат розширення знакового розряду, виконаного при автоматичному підвищенні типу змінної b до int перед операцією зсуву вправо. Нарешті, у виразі для змінної e нам вдається досягти бажаного результату - значення 0x0f. Для цього нам довелося перед зрушенням праворуч логічно помножити значення змінної b на маску 0xff, очистивши таким чином старші розряди, заповнені при автоматичному підвищенні типу. Зверніть увагу, що при цьому вже немає необхідності використовувати беззнакове зсув праворуч, оскільки ми знаємо стан знакового біта після операції AND.

3: \> java ByteUShift

b = 0xf1

b >> 4 = 0xff

b >>> 4 = 0xff

b & 0xff) >> 4 = 0x0f

Бітові оператори присвоєння

Так само, як і у випадку арифметичних операторів, у всіх бінарних бітових операторів є споріднена форма, що дозволяє автоматично надавати результат операції лівому операнду. У наступному прикладі створюються кілька змінних, з якими за допомогою операторів, зазначених вище, виконуються різні операції.

```
class OpBitEquals {

public static void main(String args[]) {

int a = 1;

int b = 2;

int c = 3;
```

```
a | = 4;

b >>= 1;

з <<= 1;

a ^= c;

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

System.out.println("b = " + b);

System.out.println("c = " + c);

} }
```

Результати виконання програми такі:

```
З:\> Java OpBitEquals

a = 3

b = 1

з = 6
```

Оператори відносини

Для того, щоб можна було порівнювати два значення, Java має набір операторів, що описують ставлення і рівність. Список таких операторів наведено у таблиці.

Оператор	Результат
==	одно
!=	не одно
>	більше

<	менше
>=	більше чи одно
<=	менше чи одно

Значення будь-яких типів, включаючи цілі та речові числа, символи, логічні значення та посилання, можна порівнювати, використовуючи оператор перевірки на рівність == та нерівність !=. Зверніть увагу - у мові Java, так само, як у C і C ++ перевірка на рівність позначається послідовністю (==). Один знак (=) – це оператор присвоєння.

Бульові логічні оператори

Булеві логічні оператори, зведення яких наведено в таблиці нижче, оперують лише з операндами типу boolean. Всі бінарні логічні оператори сприймають як операнди два значення типу boolean і повертають результат того ж типу.

Оператор	Результат	Оператор	Результат
&	логічне І (AND)	&=	І (AND) із присвоєнням
|	логічне АБО (OR)	=	АБО (OR) із присвоєнням
^	логічне виключне АБО (XOR)	^=	що виключає АБО (XOR) з привласненням
||	оператор OR швидкої оцінки виразів (short circuit OR)	==	одно
&&	оператор AND швидкої оцінки виразів (short circuit AND)	!=	не одно
!	логічне унарне заперечення (NOT)	?:	тернарний оператор if-then-else

Результати впливу логічних операторів різні комбінації значень операндов показані в таблиці.

A	B	OR	AND	XOR	NOT A
false	false	false	false	false	true
true	false	true	false	true	false
false	true	true	false	true	true
true	true	true	true	false	false

Програма, наведена нижче, майже повністю повторює вже знайомий вам приклад BitLogic. Тільки цього разу ми працюємо з булевими логічними значеннями.

```
class BoolLogic {

public static void main(String args[]) {

boolean a = true;

boolean b = false;

boolean з = a | b;

boolean d = a & b;

boolean e = a^b;

boolean f = (! a & b) | (a &! b);

boolean g =!

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

System.out.println(" b = " + b);

System.out.println(" a|b = " + c);

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

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

System.out.println("!a&b|a&!b = " + f);

System.out.println(" !a = " + g);

} }
```

```
Java BoolLogic

a = true

b = false

a | b = true

a&b = false

a^b = true

!a&b|a&!b = true

!a = false
```

Оператори швидкої оцінки логічних виразів (short circuit logical operators)

Існують два цікаві доповнення до набору логічних операторів. Це альтернативні версії операторів AND і OR, що служать для швидкої оцінки логічних виразів. Ви знаєте, якщо перший операнд оператора OR має значення true, то незалежно від значення другого операнда результатом операції буде величина true. Аналогічно у випадку оператора AND, якщо перший операнд - false, то значення другого операнда на результат не впливає - він завжди дорівнюватиме false. Якщо ви використовуєте оператори && і || замість звичайних форм & і |, то Java не робить оцінку правого операнда логічного виразу, якщо відповідь зрозуміла зі значення лівого операнда. Загальноприйнятою практикою є використання операторів && та || практично у всіх випадках оцінки булевих логічних виразів. Версії цих операторів і | застосовуються лише у бітовій арифметиці.

Тернарний оператор if-then-else

Загальна форма оператора if-then-use така:

вираз1? вираз2: вираз

Як перший операнда — «вираз1» — може бути використаний будь-який вираз, результатом якого є значення типу boolean. Якщо результат дорівнює true, виконується оператор, заданий другим операндом, тобто, «выражение2». Якщо ж перший операнд дорівнює false, то виконується третій операнд - «вираз 3». Другий і третій операнди, тобто «вираз2» та «вираз 3», повинні повертати значення одного типу і не повинні мати тип void.

У наведеній нижче програмі цей оператор використовується для перевірки дільника перед виконанням операції розподілу. У разі нульового дільника повертається значення 0.

```
class Ternary {

public static void main(String args[]) {

int a = 42;

int b = 2;

int c = 99;

int d = 0;

int e = (b == 0)? 0: (a/b);

int f = (d == 0)? 0: (c/d);

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

System.out.println("b = " + b);

System.out.println("c = " + c);

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

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

System.out.println("c/d =" + f);

} }
```

За виконання цієї програми виняткової ситуації розподілу на нуль немає і виводяться такі результаты:

```
З: java Ternary

a = 42

b = 2

з = 99

d = 0

a/b = 21

з/d = 0
```

Пріоритети операторів

У Java діє певний порядок, чи пріоритет, операцій. В елементарній алгебрі нас вчили тому, що у множення і поділу вищий пріоритет, ніж у додавання та віднімання. У програмуванні також доводиться стежити за пріоритетами операцій. У таблиці вказано у порядку зменшення пріоритети всіх операцій мови Java.

Вищий			
( )	[]	.	
~	!		
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=

==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
Нижчий			

У першому рядку таблиці наведено три незвичайні оператори, про які ми поки не говорили. Круглі дужки () використовуються для явного встановлення пріоритету. Як ви дізналися з попереднього розділу, квадратні дужки [] використовуються для індексування змінної-масиву. Оператор. (Точка) використовується для виділення елементів з посилання на об'єкт - про це ми поговоримо в главі 7. Все ж решта операторів вже обговорювалися в цьому розділі.

Явні пріоритети

Оскільки вищий пріоритет мають круглі дужки, ви завжди можете додати кілька пар дужок, якщо у вас є сумніви з приводу порядку обчислень або вам просто хочеться зробити свої код більш читабельним.

a >> b + 3

Якому із двох виразів, a >> (b + 3) або (a >> b) + 3, відповідає перший рядок? Оскільки у оператора додавання вищий пріоритет, ніж у оператора зсуву, правильна відповідь — a>> (b + a). Отже, якщо вам потрібно виконати операцію (a>>b)+ 3 без дужок не обійтися.

Оператори управління потоком виконання Java програми

Як ви знаєте, будь-який алгоритм, призначений для виконання на комп'ютері, можна розробити, використовуючи лише лінійні обчислення, розгалуження та цикли.

Будь-яка мова програмування повинна мати засоби запису алгоритмів. Вони називаються*операторами*(statements) мови. Мінімальний набір операторів повинен містити оператор для запису лінійних обчислень, умовний оператор для запису розгалуження та оператор циклу.

Зазвичай склад операторів мови програмування ширше: зручності записи алгоритмів в мову включаються кілька операторів циклу, оператор варіанти, оператори переходу, оператори опису об'єктів.

Набір операторів мови Java включає:

- Оператори опису змінних та інших об'єктів
- Оператори-вираження;
- Оператори присвоєння;
- Умовний оператор if;
- Три оператори циклу while, do-while, for;
- Оператор варіанта switch;
- Оператори переходу break, continue та return;
- Блок {};
- Порожній оператор - просто крапка з комою.

Тут наведено основний набір операторів Java.

Зауваження

У мові Java немає оператора goto.

Кожен оператор завершується крапкою з комою.

Можна поставити крапку з комою наприкінці будь-якого висловлювання, і вона стане оператором (expression statement). Але сенс це має лише для операцій присвоєння, інкременту та декременту та викликів методів. В інших випадках це марно, тому що обчислене значення виразу загубиться.

Лінійне виконання алгоритму забезпечується послідовним записом операторів. Перехід з рядка на рядок у вихідному тексті не має жодного значення для компілятора, він здійснюється лише для наочності та читання тексту.

Блок

Блок містить у собі нуль або кілька операторів з метою використовувати їх як один оператор у тих місцях, де за правилами мови можна записати лише один оператор. Наприклад, {x = 5; y =?;}. Можна записати і порожній блок, просто кілька фігурних дужок {}.

Блоки операторів часто використовуються для обмеження області дії змінних і просто для покращення читання тексту програми.

Оператори присвоєння

Крапка з комою наприкінці будь-якої операції присвоювання перетворює її на оператор присвоювання. Побічна дія операції – привласнення – стає в операторі основною.

Різниця між операцією та оператором присвоєння носить лише теоретичний характер. Привласнення найчастіше використовується як оператор, а не операція.

Умовний оператор

Умовний оператор (if-then-else statement) у мові Java записується так:

if (логВир) оператор1 else оператор2

і діє в такий спосіб. Спочатку обчислюється логічний вираз*логвир.*Якщо результат true, то діє*оператор!*і на цьому дія умовного оператора завершується,*оператор2*не діє, далі виконуватиметься наступний за if оператор. Якщо результат false, то діє*оператор2,*при цьому оператор! взагалі не виконується.

Умовний оператор може бути скороченим (if-then statement):

if (логВир) оператор!

і у разі false не виконується нічого.

Синтаксис мови не дозволяє записувати кілька операторів ні в гілки then, ні в гілки else. За потреби складається блок операторів у фігурних дужках. Угоди "Code Conventions" рекомендують завжди використовувати фігурні дужки та розміщувати оператор на кількох рядках з відступами, як у наведеному нижче прикладі:

```
if (a <x) {

   x = a + b;
}
else {

   x = a - b;

}
```

Це полегшує додавання операторів до кожної галузі при зміні алгоритму. Ми не суворо дотримуватимемося цього правила, щоб не збільшувати обсяг книги.

Дуже часто одним із операторів є знову умовний оператор, наприклад:

```
if (п == 0) {

    sign = 0;

} else if (n < 0) {

   sign = -1;

} else { sign = 1;

}
```

При цьому може виникнути така ситуація (dangling else):

```
int ind = 5, x = 100;

if (ind >= 10) if (ind <= 20) x = 0; else x = 1;
```

Збереже змінна x значення 0 чи дорівнює 1? Тут необхідне вольове рішення, і загальне більшість мов, зокрема і Java,. правило таке: гілка else відноситься до найближчого зліва услдвію if, що не має своєї гілки else. Тому в прикладі змінна x залишиться рівною 0.

Змінити цей порядок можна за допомогою блоку:

```
if (ind > 10) {if (ind < 20) x = 0; else x = 1;}
```

Загалом не варто захоплюватися складними вкладеними умовними операторами. Перевірки умов займають багато часу. По можливості краще використовувати логічні операції, наприклад, у прикладі можна написати

```
if (ind >= 10 && ind <= 20) x = 0; else x = 1;
```

У наведеному нижче лістингу обчислюється коріння квадратного рівняння ax 2 + bx + c = 0 для будь-яких коефіцієнтів, у тому числі і нульових.

Лістинг Обчислення коренів квадратного рівняння

```
class QuadraticEquation{

public static void main(String[] args){

double a = 0.5, Ъ = -2.7, з = 3.5, d, eps = le-8;

if (Math.abs(a) < eps)

if (Math.abs(b) < eps)

if (Math.abs(c) < eps) // Усі коефіцієнти дорівнюють нулю

System.out.println("Рішення - будь-яке число");
```

```
else
System.out.println("Рішень немає");
else
System.out.println("xl = x2 = " +(-c / b) ) ;
else { // Коефіцієнти не дорівнюють нулю
if((d = b**b — 4*a*c)< 0.0){ // Комплексне коріння
d = 0.5 * Math.sqrt (-d) / a;
a = -0.5 * b/a;
System.out.println("xl = " +a+ " +i " +d+
", x2 = "+a+"-i "+d);
} else {
// Речовинні коріння
d = 0.5 * Math.sqrt (d) / a;
a = -0.5*b/a;
System.out.println("x1="+(a+d)+", x2="+(a-d));
}
}
)
```

}

У цій програмі використані методи обчислення модуля abs() і квадратного кореня sqrt про речовинне число з вбудованого в Java API класу Math. Оскільки всі обчислення з речовими числами виробляються приблизно, ми вважаємо, що коефіцієнт рівняння дорівнює нулю, якщо його модуль менше 0,00000001. Зверніть увагу на те, як у методі println використовується зчеплення рядків, і на те, як операція присвоювання при обчисленні дискримінанта вкладена в логічний вираз.

Оператори циклу

Основний оператор циклу – оператор while – виглядає так:

while(логВир) оператор

Спочатку обчислюється логічний вираз*логВир;*якщо його значення true, то виконується оператор, що утворює цикл. Потім знову обчислюється*лог-вир*і діє оператор, і так доти, доки не вийде значення false. Якщо*логВир*спочатку дорівнює false, то*оператор*не буде виконано жодного разу. Попередня перевірка забезпечує безпеку виконання циклу, дозволяє уникнути переповнення, розподілу на нуль та інших неприємностей. Тому оператор while є основним, а деяких мовами і єдиним оператором циклу.

Оператор у циклі може бути порожнім, наприклад, наступний фрагмент коду:

```
int i = 0;

double s = 0.0;

while ((s += 1.0 / ++i) < 10);
```

обчислює кількість і додань, які необхідно зробити, щоб гармонійна сума s досягла значення 10. Такий стиль характерний для мови C. Не варто їм захоплюватися, щоб не перетворити текст програми на шифрування, на яке ви самі через пару тижнів дивитиметеся з подивом.

Можна організувати і нескінченний цикл:

```
while (true) оператор
```

Звичайно, з такого циклу слід передбачити якийсь вихід, наприклад оператором break, як у лістингу 1.5. В іншому випадку програма зациклиться, і вам доведеться припиняти її виконання "комбінацією з трьох пальців" <Ctrl>+<Alt>+<Del> у MS Windows 95/98/ME, комбінацією <Ctrl>+<C> у UNIX або через Task Manager у Windows NT/2000.

Якщо в цикл треба включити кілька операторів, слід утворити блок операторів {}.

Другий оператор циклу – оператор do-while – має вигляд do оператор while*(логВир)*

Тут спочатку виконується оператор, а потім відбувається обчислення логічного виразу логвир. Цикл виконується, поки логвир залишається рівним true.

Істотна відмінність між цими двома операторами циклу лише тому, що у циклі do-while оператор обов'язково виконається хоча б один раз.

Наприклад, нехай задана якась функція f(x), що має на відрізку [a, b] рівно один корінь. У лістингу наведена програма, яка обчислює цей корінь приблизно методом поділу навпіл (бісекції, дихотомій).

Лістинг Знаходження кореня нелінійного рівняння методом бісекції

```
class Bisection{

static double f(double x) {

return x * x * x - 3 * x * x +3; // Або щось інше

}

public static void main(String!] args) {

double a = 0.0, b = 1,5, c, y, eps = le-8;

do{

з = 0.5 * (a + b); y = f(c);

if (Math.abs(y) < eps) break;
```

```
// Корінь знайдено. Виходимо із циклу

// Якщо кінцях відрізка [a; з]

// Функція має різні знаки:

if (f(a) * y < 0.0) b = c;

// Отже, корінь тут. Переносимо точку b у точку з

/ / Інакше:

else a*c;

// Переносимо точку а в точку з

// Продовжуємо, поки відрізок [a; Ь] не стане малий

} while (Math, abs (ba)> = eps);

System.out.println("x = "+c+", f("+c+") = "+y);

}

}
```

Клас Bisection складніший за попередні приклади: у ньому крім методу main () є ще метод обчислення функції f(x). Тут метод f дуже простий: він обчислює значення многочлена і повертає його як значення функції, причому все це виконується одним оператором:

return вираз

У методі main про з'явився ще один новий оператор break, який просто припиняє виконання циклу, якщо ми завдяки щасливому випадку натрапили на наближене значення кореня. Уважний читач помітив появу модифікатора static в оголошенні методу f(). Він необхідний тому, що метод f про викликається зі статичного методу main o.

Третій оператор циклу – оператор for – виглядає так:

for (списокВир ; логНьр; слисокВир2) оператор

Перед виконанням циклу обчислюється список виразів*СписокВир1.*Це нуль або кілька виразів, перерахованих через кому. Вони обчислюються ліворуч, і в наступному виразі вже можна використовувати результат попереднього виразу. Як правило, тут задаються початкові значення змінного циклу.

Потім обчислюється логічний вираз логвір. Якщо воно істинно, true, то діє оператор, потім обчислюються зліва направо вирази зі списку виразів*СписокВир2.*Далі знову перевіряється*логвир.*Якщо воно істинно, то виконується оператор і*списокВир2*і т. д. Як тільки логйир стане рівним false, виконання циклу закінчується.

Коротше кажучи, виконується послідовність операторів

список Вир1; while (логВир) {

оператор

слисокВир2; }

з тим винятком, що якщо оператором у циклі є оператор

continue, то список_вир2 все-таки виконується.

Замість*списокВир1*може стояти одне визначення змінних обов'язково з початковим значенням. Такі змінні відомі лише у межах цього циклу.

Будь-яка частина оператора for може бути відсутня: цикл може бути порожнім, вирази в заголовку теж, при цьому точки з комою зберігаються. Можна задати нескінченний цикл:

for (;;) оператор

І тут у тілі циклу слід передбачити якийсь вихід.

Хоча в операторі for закладені великі можливості, використовується він, головним чином, для перерахувань, коли їх число відоме, наприклад, фрагмент коду

```
int s = 0;

for (int k = 1; k <= N; k ++) s + = k * k;

// Тут змінна k вже невідома
```

обчислює суму квадратів перших N натуральних чисел.

Оператор continue та мітки

Оператор continue використовується лише в операторах циклу. Він має дві форми. Перша форма складається лише зі слова continue та здійснює негайний перехід до наступної ітерації циклу. У черговому фрагменті коду оператор continue дозволяє обійти поділ на нуль:

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

if (i '== j) continue;

s += 1.0/(i - j);

}
```

Друга форма містить мітку:

continue мітка

*мітка*записується, як усі ідентифікатори, з літер Java, цифр і символу підкреслення, але не вимагає жодного опису. Мітка ставиться перед оператором або фігурною дужкою, що відкриває, і відокремлюється від них двокрапкою. Так виходить*помічений оператор*або*помічений блок.*

Друга форма використовується тільки у разі кількох вкладених циклів для негайного переходу до чергової ітерації одного з об'ємних циклів, а саме поміченого циклу.

Оператор break

Оператор break використовується в операторах циклу та операторі варіанта для негайного виходу з цих конструкцій.

Оператор break мітка

застосовується всередині помічених операторів циклу, оператора варіанта або позначеного блоку для негайного виходу цих операторів. Наступна схема пояснює цю конструкцію.

MI: { // Зовнішній блок

M2: { // Вкладений блок - другий рівень

M3: { // Третій рівень вкладеності...

if (щось сталося) break M2;

// Якщо true, то тут нічого не виконується

}

// Тут також нічого не виконується

}

// Сюди передається управління

}

Спочатку збиває з пантелику та обставина, що мітка ставиться перед блоком чи оператором, а управління передається цей блок чи оператор. Тому не варто захоплюватися оператором break із міткою.

Оператор варіанта

Оператор варіанта switch організує розгалуження за декількома напрямками. Кожна гілка відзначається константою або константним виразом будь-якого цілого типу (крім long) і вибирається, якщо значення певного виразу збігатиметься з цією константою. Уся конструкція виглядає так.

switch (цілВир) {

case констВир1: оператор1

case констВир2: оператор2

. . . . .

case констВирN: операторN

default: операторDef

}

Вираз, що стоїть у дужках.*цілвир*може бути типу byte, short, int, char, але з long. Цілі числа або цілочисленні вирази, складені з констант,*констВир*теж повинні мати тип long.

Оператор варіанта виконується так. Всі константні вирази обчислюються заздалегідь, на етапі компіляції, і повинні мати відмінні значення. Спочатку обчислюється цілочислове вираз; цілир. Якщо. ОНО збігається з однією з констант, виконується оператор, відзначений цією константою. Потім виконуються ("fall through labels") всі наступні оператори, включаючи і*операторОе£,*і робота оператора варіайтузакінчується.

Якщо ж жодна константа не дорівнює значенню виразу, то виконується*операторОе£*і всі наступні за ним оператори. Тому гілка default повинна записуватися останньою. Гілка default може бути відсутня, тоді в цій ситуації оператор варіанта взагалі нічого не робить.

Таким чином, константи у варіантах case грають роль тільки міток, точок входу в оператор варіанта, а далі виконуються всі оператори, що залишилися, в порядку їх запису.

Знавцям Pascal

Після виконання одного варіанту оператор switch продовжує виконувати всі варіанти, що залишилися.

Найчастіше необхідно "пройти" лише одну гілку операторів. У такому випадку використовується оператор break, який відразу ж припиняє виконання оператора switch. Може знадобитися виконати той самий оператор у різних гілках case. У цьому випадку ставимо кілька позначок case поспіль. Ось простий приклад.

```
switch(dayOfWeek){

case 1: case 2: case 3: case 4: case 5:

System.out.println("Week-day");, break;

case 6: case 7:

System.out.println("Week-end"); break;

default:

System.out.printlnt"Unknown day");

}
```

Зауваження

Не забувайте завершувати варіанти оператором break.

Масиви

Як завжди у програмуванні*масив*- Це сукупність змінних одного типу, що зберігаються в суміжних осередках оперативної пам'яті.

Масиви в мові Java ставляться до типів посилань і описуються своєрідно, але характерно для типів посилань. Опис проводиться у три етапи.

Перший етап -*оголошення*(Declaration). На цьому етапі визначається лише змінна типу*посилання*(Reference)*на масив,*містить тип масиву. Для цього записується ім'я типу елементів масиву, квадратними дужками вказується, що оголошується посилання на масив, а не проста змінна, і перераховуються імена змінних типу посилання, наприклад,

```
double[] a, b;
```

Тут визначено дві змінні - посилання а і ь на масиви типу double. Можна поставити квадратні дужки безпосередньо після імені. Це зручно робити серед визначень звичайних змінних:

int I = 0, ar [], k = -1;

Тут визначено дві змінні цілого типу і і k, і оголошено посилання на цілий масив аг.

Другий етап -*визначення*(Installation). На цьому етапі вказується кількість елементів масиву, що його називають*довжиною,*виділяється місце для масиву в оперативній пам'яті, змінна-посилання отримує адресу масиву. Всі ці дії здійснюються ще однією операцією мови Java - операцією new*тип,*виділяє ділянку в оперативній пам'яті для об'єкта зазначеного в операції типу і повертає як результат адресу цієї ділянки. Наприклад,

a = new double [5];

b = New double [100];

ar = new int [50];

Індекси масивів завжди починаються з 0. Масив а складається з п'яти змінних а[0], а[1], , а[4]. Елементу а[5] у масиві немає. Індекси можна задавати будь-якими цілими виразами, крім типу long, наприклад, а[i+j], а[i%5], а[++i]. Виконуюча система Java слідкує за тим, щоб значення цих виразів не виходили за межі довжини масиву.

Третій етап -*ініціалізація*(Initialization). На цьому етапі елементи масиву набувають початкових значень. Наприклад,

a [0] = 0.01; a [1] = -3.4; a [2] = 2: 89; a [3] = 4.5; a [4] = -6.7;

for (int i = 0; i < 100; i + +) b [i] = 1.0 / i;

for (int i = 0; i < 50; i + +) ar [i] = 2 * i + 1;

Перші два етапи можна поєднати:

doublet] a = новий double [5], b = новий double [100];

int i = 0, ar [] = new int [50], k = -1;

Можна відразу задати і початкові значення, записавши їх у фігурних дужках через кому у вигляді констант або константних виразів. При цьому навіть необов'язково вказувати кількість елементів масиву, воно дорівнюватиме кількості початкових значень;

double[] a = {0.01, -3.4, 2.89, 4.5, -6.7};

Можна поєднати другий та третій етап:

a = new doublet] {0.1, 0.2, -0.3, 0.45, -0.02};

Можна навіть створити безіменний масив, відразу ж використовуючи результат операції new, наприклад, так:

System.out.println(new char[] {'H', 'e', '1', '1', 'o'});

Посилання на масив не є частиною описаного масиву, його можна перекинути на інший масив того ж типу операцією присвоєння. Наприклад, після присвоєння a = ь обидві посилання a і ь вказують на один і той же масив зі 100 речових змінних типу double і містять одну і ту ж адресу.

Посилання може присвоїти "порожнє" значення null, що не вказує на жодну адресу оперативної пам'яті:

ar = null;

Після цього масив, на який вказувало це посилання, втрачається, якщо на нього не було інших посилань.

Крім простої операції присвоювання, з посиланнями можна робити тільки порівняння на рівність, наприклад, a = ь, і нерівність, a != b. При цьому зіставляються адреси, що містяться в посиланнях, ми можемо дізнатися, чи вони посилаються на один і той же масив.

Зауваження - Масиви Java завжди визначаються динамічно, хоча посилання на них задаються статично.

Крім посилання на масив, для кожного масиву автоматично визначається ціла константа з тим самим ім'ям length. Вона дорівнює довжині масиву. До кожного масиву ім'я цієї константи уточнюється ім'ям масиву через точку. Так, після наших визначень константа a.length дорівнює 5, константа b. length дорівнює 100, а ar. length дорівнює 50.

Останній елемент масиву можна записати так: a [a. length - 1], передостанній - a [a. length - 2] і т. д. Елементи масиву зазвичай перебираються у циклі виду:

double a Min = a [0], a Max = a Min;

for (int i = 1; i < a.length; i++){

```
if <a[i] < aMin) aMin = a[i];
if (a[i] > aMax) aMax = a[i];
}
double range = aMax - aMin;
```

Тут обчислюється діапазон значень масиву.

Елементи масиву - це звичайні змінні свого типу, з ними можна виконувати всі операції, допустимі для цього типу:

(a[2] + a[4]) / a[0] тощо.

Багатовимірні масиви

Елементами масивів Java можуть бути знову масиви. Можна оголосити:

```
char[] [] c;
```

що еквівалентно

```
char з [] з [];
```

або

```
char з [] [];
```

Потім визначаємо зовнішній масив:

```
з = new char[3][];
```

Стає ясно, що з масив, що складається з трьох елементів-масивів. Тепер визначаємо його елементи-масиви:

```
з [0] = new char [2];
```

c[1] = new char[4];

c[2] = new char[3];

Після цих визначень змінна c.length дорівнює з, c[0] .length дорівнює 2,

c[l].length дорівнює 4 і з[2.length дорівнює 3.

Нарешті, задаємо початкові значення з [0] [0] = 'a', з [0] [1] = 'r',

c[1][0] = 'r',c[1][1] = 'a',c[1][2] = 'y' і т.д.

Зауваження

Двовимірний масив Java не повинен бути прямокутним.

Описи можна скоротити:

int[] [] d = new int[3] [4];

А початкові значення поставити так:

int[][] inds = {{I, 2, 3}, {4, 5, 6}};

У лістингу наведено приклад програми, що обчислює перші 10 рядків трикутника Паскаля, що заносить їх у трикутний масив та виводить його елементи на екран. Мал. показує виведення цієї програми.

Лістинг Трикутник Паскаля

class PascalTriangle{

public static final int LINES = 10; // Так визначаються констан

public static void main(String[] args) {

int[][] p, = new int [LINES] [];

```
p[0] = new int[1];
System, out. println (p[0][0]=1);
p[l] = new int[2];
p[l][0] = p[l][1] = 1;
System.out.println(p[1][0] + " " + p[l][l]);
for (int i = 2; i < LINES; i++){
p[i] = new int[i+l];
System.out.print((p[i][0] = 1) + " ");
for (int j = 1; j < i; j++)
System.out. print ((p[i][j] = p[il][jl]-bp[il][j]) + "");
System, out. println (p [i] [i] = 1)
}
}
}
```

Як можна «отримати» «випадкові» числа:

```
Random rand = новий Random();
  z[i]=(int) (100 * rand.nextDouble());
```

Робота з рядками

У мовах С немає вбудованої підтримки такого об'єкта, як рядок. У них при необхідності передається адреса послідовності байтів, вміст яких трактується як символи до тих пір, поки не буде зустрінуто нульовий байт, що позначає кінець рядка. У пакет java.lang вбудований клас, що інкапсулює структуру даних, що відповідає рядку. Цей клас, званий String, нічим іншим, як об'єктне уявлення незмінного символьного масиву. У цьому класі є методи, які дозволяють порівнювати рядки, здійснювати пошук і витягувати певні символи і підрядки. Клас StringBuffer використовується тоді, коли рядок після створення потрібно змінювати.

УВАГА

І String, і StringBuffer оголошені final, що означає, що жодного з цих класів не можна робити підкласи. Це було зроблено для того, щоб можна було застосувати деякі види оптимізації, що дозволяють збільшити продуктивність при виконанні операцій обробки рядків.

Конструктори

Як і у будь-якого іншого класу, можна створювати об'єкти типу String за допомогою оператора new. Для створення порожнього рядка використовується конструктор без параметрів:

```
String s = new String():
```

Наведений нижче фрагмент коду створює об'єкт s типу String ініціалізуючи його рядком з трьох символів, переданих конструктору як параметр символьному масиві.

```
char chars[] = { 'a', 'b', 'c'}:

String s = new String(chars);

System.out.println(s):
```

Цей фрагмент коду виводить рядок "abc". Отже, цей конструктор має 3 параметри:

```
String (char chars [], int початковий Індекс, int число Символів);
```

Використовуємо такий спосіб ініціалізації у нашому черговому прикладі:

```
char chars[] = { 'a', 'b', 'c', 'd', 'e', 'f'}:
```

```
String s = new String (chars, 2, 3);

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

Цей фрагмент виведе cde.

Спеціальний синтаксис для роботи з рядками

Java містить кілька приємних синтаксичних доповнень, мета яких — допомогти програмістам у виконанні операцій з рядками. Серед таких операцій створення об'єктів типу String злиття кількох рядків та перетворення інших типів даних на символьне уявлення.

Створення рядків

Java включає стандартне скорочення для цієї операції - запис у вигляді літерала, в якій вміст рядка полягає в пару подвійних лапок. Нижче наведений фрагмент коду еквівалентний одному з попередніх, в якому рядок ініціалізувалася масивом типу char.

```
String s = "abc";

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

Один із загальних методів, які використовуються з об'єктами String - метод length, що повертає число символів у рядку. Черговий фрагмент виводить число 3, оскільки в рядку, що використовується в ньому, — 3 символи.

```
String s = "abc";

System.out.println(s.length);
```

У Java цікаво те, що для кожного рядка-літералу створюється свій представник класу String, так що ви можете викликати методи цього класу безпосередньо з рядками-літералами, а не лише з перемінними посиланнями. Ще один приклад також виводить число 3.

```
System.out.println("abc".Length());
```

Злиття рядків

Рядок

```
String s = "He is" + age + "years old.";
```

в якій за допомогою оператора + три рядки об'єднуються в один, прочитати і зрозуміти безумовно легше, ніж її еквівалент, записаний з явними викликами тих методів, які неявно були використані в першому прикладі:

```
String s = new StringBuffer("He is ").append(age);

s.append(" years old.").toString();
```

За визначенням, кожен об'єкт класу String не може змінюватися. Не можна ні вставити нові символи в існуючий рядок, ні поміняти в ній одні символи на інші. І додати один рядок на кінець іншого теж не можна. Тому транслятор Java перетворює операції, що виглядають, як модифікація об'єктів String, в операції із спорідненим класом StringBuffer.

ЗАУВАЖЕННЯ

Все це може здатися вам невиправдано складним. А чому не можна обійтися одним класом String, дозволивши йому поводитися приблизно так само, як StringBuffer? Вся справа у продуктивності. Той факт, що об'єкти типу String Java незмінні, дозволяє транслятору застосовувати до операцій з ними різні способи оптимізації.

Послідовність виконання операторів

Давайте ще раз звернемося до нашого останнього прикладу:

```
String s = "He is" + age + " years old.";
```

У тому випадку, коли age - не String, а змінна, скажімо, типу int, у цьому рядку коду укладено ще більше магії транслятора. Ціле значення змінної int передається суміщеному методу append класу StringBuffer, який перетворює його в текстовий вигляд і додає в кінець рядка, що міститься в об'єкті. Вам потрібно бути уважним при спільному використанні цілих виразів і злиття рядків, інакше результат може вийти зовсім не на той, на який ви чекали. Погляньте на наступний рядок:

```
String s = "four:" + 2 + 2;
```

Можливо, ви сподіваєтеся, що s буде записаний рядок «four: 4»? Не вгадали — з вами зіграла злий жарт послідовність виконання операторів. Отже, в результаті виходить "four: 22".

Для того, щоб першим виконалося складання цілих чисел, потрібно використовувати дужки:

```
String s = "four:" + (2 + 2);
```

Перетворення рядків

У кожному класі String є метод toString — або власна реалізація, або варіант за умовчанням, успадкований від класу Object. Клас у нашому черговому прикладі заміщає успадкований метод toStrring своїм власним, що дозволяє йому виводити значення змінних об'єкта.

```
class Point {

int x, y;

Point(int x, int y) {

this.x = x;

this.y = y;

}

public String toString() {

return "Point["+x+", "+y+"]";

} }

class toStringDemo {

public static void main(String args[]) {
```

```
Point p = new Point (10, 20);

System.out.println("p = " + p);

} }
```

Нижче наведено результат, отриманий під час запуску цього прикладу.

```
3:\> Java toStringDemo

p = Point[10, 20]
```

Вилучення символів

Щоб витягти одиночний символ з рядка, ви можете послатися безпосередньо на індекс символу в рядку за допомогою методу charAt. Якщо ви хочете в один прийом отримати кілька символів, можете скористатися методом getChars. У наведеному нижче фрагменті показано, як витягувати масив символів з об'єкта типу String.

```
class getCharsDemo {

public static void main(String args[]) {

String s = "Цей є Demo getChars method.";

int start = 10;

int end = 14;

char buf [] = New char [end - start];

s.getChars(start, end, buf, 0);

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

```
} }
```

Зверніть увагу – метод getChars не включає у вихідний буфер символ із індексом end. Це добре видно з висновку нашого прикладу — рядок складається з 4 символів.

3:\> java getCharsDemo

```
demo
```

Для зручності роботи в String є ще одна функція - toCharArray, яка повертає у вихідному масиві типу char весь рядок. Альтернативна форма того самого механізму дозволяє записати вміст рядка в масив типу byte, при цьому значення старших байтів в 16-бітних символах відкидаються. Відповідний метод називається getBytes, і його параметри мають той самий сенс, що й параметри getChars, але з єдиною різницею - як третій параметр треба використовувати масив типу byte.

Порівняння

Якщо ви хочете дізнатися, чи однакові два рядки, вам слід скористатися методом equals класу String. Альтернативна форма цього методу називається equalsIgnoreCase, при її використанні відмінність регістрів букв у порівнянні не враховується. Нижче наведено приклад, що ілюструє використання обох методів:

```
class equalDemo {

public static void main(String args[]) {

String s1 = "Hello";

String s2 = "Hello";

String s3 = "Good-bye";
```

```
String s4 = "HELLO";

System.out.println(s1 + "equals" + s2 + "->" + s1.equals(s2));

System.out.println(s1 + "equals" + s3 + "->" + s1.equals(s3));

System.out.println(s1 + "equals" + s4 + "->" + s1.equals(s4));

System.out.println(s1 + "equalsIgnoreCase" + s4 + "->" +

                   s1.equalsIgnoreCase(s4));

} }
```

Результат запуску цього прикладу:

З:\> java equalsDemo

```
Hello equals Hello -> true

Hello equals Good-bye -> false

Hello equals HELLO -> false

Hello equalsIgnoreCase HELLO -> true
```

У класі String реалізовано групу сервісних методів, що є спеціалізованими версіями методу equals. Метод regionMatches використовується для порівняння підрядка у вихідному рядку з підрядком у рядку-параметрі. Метод startsWith перевіряє, чи починається дана підрядка фрагментом, переданим методу як параметр. Метод endsWith перевіряє, чи збігається з параметром кінець рядка.

Рівність

Метод equals і оператор == виконують дві абсолютно різні перевірки. Якщо метод equal порівнює символи всередині рядків, то оператор == порівнює дві змінні-посилання на об'єкти і перевіряє, чи вказують вони на різні об'єкти або на той самий. У нашому прикладі це добре видно - вміст двох рядків однаково, але, тим не менш, це - різні об'єкти, так що equals і == дають різні результати.

```
class EqualsNotEqualTo {

public static void main(String args[]) {

String s1 = "Hello";

String s2 = новий String(s1);

System.out.println(s1 + "equals" + s2 + "->" + s1.equals(s2));

System.out.println(s1 + " == " + s2 + ", -> " + (s1 == s2));

} }
```

Ось результат запуску цього прикладу:

C:\> java EqualsNotEqualTo

Hello equals Hello -> true

Hello == Hello -> false

Упорядкування

Найчастіше буває недостатньо просто знати, чи є два рядки ідентичними. Для додатків, у яких потрібно сортування, потрібно знати, який із двох рядків менший за інший. Для відповіді це питання потрібно скористатися методом compareTo класу String. Якщо ціле значення, повернене методом, негативне, то рядок, з якого був викликаний метод, менший за рядок-параметр, якщо позитивно — більший. Якщо ж метод compareTo повернув значення 0, рядки ідентичні. Нижче наведено програму, у якій виконується бульбашкова сортування масиву рядків, а порівняння рядків використовується метод compareTo. Ця програма видає відсортований у алфавітному порядку список рядків.

```
class SortString {
static String arr[] = {"Now", "is", "the", "time", "for", "all",
                       "good", "men", "to", "come", "to", "the",
                       "aid", "of", "their", "country"};
public static void main(String args[]) {
for (int j = 0; i < arr.length; j++) {
for (int i = j + 1; i < arr.length; i++) {
          if (arr[i].compareTo(arr[j]) < 0) {
              String t = arr [j];
              arr[j] = arr[i];
              arr[i] = t;
          }
     }
     System.out.println(arr[j]);
}
} }
```

indexOf та lastIndexOf

У клас String включена підтримка пошуку певного символу або підрядка, для цього в ньому є два методи - indexOf та lastIndexOf. Кожен з цих методів повертає індекс того символу, який ви хотіли знайти, або індекс початку підстроки, що шукається. У будь-якому випадку, якщо пошук виявився невдалим, методи повертають значення -1. У прикладі показано, як користуватися різними варіантами цих методів пошуку.

```
class indexOfDemo {

public static void main(String args[]) {

String s = "Now is the time for all good men " +

           "To come to the aid of their country " +

           "and pay their due taxes.";

System.out.println(s);

System.out.println("indexOf(t) = " + s.indexOf('f'));

System.out.println("lastIndexOf(t) = " + s.lastIndexOf('f'));

System.out.println("indexOf(the) = "+ s.indexOf("the"));

System.out.println("lastIndexOf(the) = "+ s.lastIndexOf("the"));

System.out.println("indexOf(t, 10) = " + s.indexOf('f' , 10));

System.out.println("lastIndexOf(t, 50) = " + s.lastIndexOf('f' , 50));

System.out.println("indexOf(the, 10) = "+ s.indexOf("the", 10));

System.out.println("lastIndexOf(the, 50) = "+ s.lastIndexOf("the", 50));

} }
```

Нижче наведено результат роботи цієї програми. Зверніть увагу на те, що індекси в рядках починаються з нуля.

```
3:> java indexOfDemo

Now is the time for all good men to come to aid of their country

and pay їx due taxes.

indexOf(t) = 7

lastlndexOf(t) = 87

indexOf(the) = 7

lastlndexOf(the) = 77

index0f(t, 10) = 11

lastlndex0f(t, 50) = 44

index0f(the, 10) = 44

lastlndex0f(the, 50) = 44
```

Модифікація рядків під час копіювання

Оскільки об'єкти класу String не можна змінювати, кожного разу, коли вам захочеться модифікувати рядок, доведеться або копіювати її в об'єкт типу StringBuffer, або використовувати один з описаних нижче методів класу String, які створюють нову копію рядка, вносячи до неї зміни.

substring

Ви можете вилучити підрядок з об'єкта String, використовуючи метод substring. Цей метод створює нову копію символів з діапазону індексів оригінального рядка, який ви вказали під час виклику. Можна вказати лише індекс першого символу потрібного підрядка - тоді будуть скопійовані всі символи, починаючи з зазначеного до кінця рядка. Також можна вказати і початковий, і кінцевий індекси - при цьому в новий рядок будуть скопійовані всі символи, починаючи з першого вказаного, і до (але не включаючи) символу, заданого кінцевим індексом.

"Hello World". substring(6) -> "World"

"Hello World". substring(3,8) -> "lo Wo"

concat

Злиття або конкатенація рядків виконується за допомогою методу concat. Цей метод створює новий об'єкт String, копіюючи вміст вихідного рядка і додаючи в її кінець рядок, вказаний у параметрі методу.

"Hello".concat("World") -> "Hello World"

replace

Методу replace як параметри задаються два символи. Усі символи, що збігаються з першим, замінюються на нову копію рядка на другий символ.

"Hello".replace('l', 'w') -> "Hewwo"

доLowerCase і toUpperCase

Ця пара методів перетворює всі символи вихідного рядка на нижній і верхній регістр, відповідно.

"Hello".toLowerCase() -> "hello"

"Hello".toUpperCase() -> "HELLO"

trim

І, нарешті, метод trim прибирає з вихідного рядка усі провідні та замикаючі прогалини.

"Hello World ".trirn() -> "Hello World"

valueOf

Якщо ви маєте справу з будь-яким типом даних і хочете вивести значення цього типу в зручному для читання вигляді, спочатку доведеться перетворити це значення в текстовий рядок. Для цього існує метод ValueOf. Такий статичний метод визначено для будь-якого існуючого Java типу даних (всі ці методи поєднані, тобто використовують одне і те ж ім'я). Завдяки цьому не важко перетворити на рядок значення будь-якого типу.

StringBuffer

StringBuffer - близнюк класу String, що надає багато з того, що зазвичай потрібно при роботі з рядками. Об'єкти класу String є рядками фіксованої довжини, які не можна змінювати. Об'єкти типу StringBuffer є послідовністю символів, які можуть розширюватися і модифікуватися. Java активно використовує обидва класи, але багато програмістів вважають за краще працювати тільки з об'єктами типу String, використовуючи оператор +. При цьому Java виконує всю необхідну роботу із StringBuffer за сценою.

Конструктори

Об'єкт StringBuffer можна створити без параметрів, при цьому в ньому буде зарезервовано місце розміщення 16 символів без можливості зміни довжини рядка. Ви також можете передати конструктору ціле число для того, щоб явно задати необхідний розмір буфера. І, нарешті, ви можете передати конструктору рядок, при цьому він буде скопійований в об'єкт і додатково до цього в ньому буде зарезервовано ще для 16 символів. Поточну довжину StringBuffer можна визначити, викликавши метод length, а визначення всього місця, зарезервованого під рядок в об'єкті StringBuffer потрібно скористатися методом capacity. Нижче наведено приклад, який пояснює це:

```
class StringBufferDemo {

public static void main(String args[]) {
```

```
StringBuffer sb = New StringBuffer("Hello");

System.out.println("buffer="+sb);

System.out.println("length=" + sb.length());

System.out. println("capacity =" + sb.capacity());

} }
```

Ось висновок цієї програми, з якого видно, що в об'єкті String-Buffer для маніпуляцій із рядком зарезервовано додаткове місце.

З:\> java StringBufferDemo

```
buffer = Hello

length = 5

capacity = 21
```

ensureCapacity

Якщо після створення об'єкта StringBuffer захочете зарезервувати в ньому місце для певної кількості символів, ви можете для встановлення розміру буфера скористатися методом ensureCapacity. Це буває корисно, коли ви знаєте, що вам доведеться додавати до буфера багато невеликих рядків.

setLength

Якщо вам раптом знадобиться явно встановити довжину рядка в буфері, скористайтеся методом setLength. Якщо ви задасте значення, більше ніж довжина рядка, що міститься в об'єкті, цей метод заповнить кінець нового, розширеного рядка символами з кодом нуль. У наведеній трохи далі програмі setCharDemo метод sstLength використовується для укорочування буфера.

charAt i setCharAt

Одиночний символ може бути вилучений із об'єкта StringBuffer за допомогою методу charAt. Інший метод setCharAt дозволяє записати в задану позицію рядка потрібний символ. Використання цих методів проілюстровано в прикладі:

```
class setCharAtDemo {

public static void main(String args[]) {

StringBuffer sb = New StringBuffer("Hello");

System.out.println("buffer before =" + sb);

System.out.println("charAt(1) before = " + sb.charAt(1));

sb.setCharAt(1, 'i');

sb.setLength(2);

System.out.println("buffer after="+sb);

System.out.println("charAt(1) after = " + sb.charAt(1));

} }
```

Ось висновок, отриманий під час запуску цієї програми.

C:\> java setCharAtDemo

```
buffer before = Hello

charAt(1) before = e
```

```
buffer after = Hi

charAt(1) after = i
```

append

Метод append класу StringBuffer зазвичай викликається неявно при використанні оператора + у виразах рядків. Для кожного параметра викликається метод String.valueOf, і його результат додається до поточного об'єкта StringBuffer. До того ж, при кожному виклику метод append повертає посилання на об'єкт StringBuffer, з яким він був викликаний. Це дозволяє вибудовувати в ланцюжок послідовні виклики методу, як показано в черговому прикладі.

```
class appendDemo {

public static void main(String args[]) {

String s;

int a = 42;

StringBuffer sb = новий StringBuffer(40);

s = sb.append("a = ").append(a).append("!").toString();

System.out.println(s);

} }
```

Ось висновок цього прикладу:

З:\> Java appendDemo

```
a = 42!
```

insert

Метод insert ідентичний методу append тому, що для кожного можливого типу даних існує своя суміщена версія цього методу. Правда, на відміну від append, він не додає символи, що повертаються методом String.valueOf, в кінець об'єкта StringBuffer, а вставляє їх у певне місце в буфері, яке задається першим його параметром. У черговому прикладі рядок "there" вставляється між "hello" і "world!".

```
class insertDemo {

public static void main(String args[]) {

StringBuffer sb = new StringBuffer("hello world!");

sb.insert(6,"there ");

System.out.println(sb);

} }
```

Під час запуску ця програма виводить наступний рядок:

3:\> java insertDemo

```
hello there world!
```

Без рядків не обійдешся

Майже будь-який аспект програмування Java на якомусь етапі передбачає використання класів String і StringBuffer. Вони знадобляться і при налагодженні, і при роботі з текстом, і при вказівці імен файлів та адрес URL як параметрів методів. Кожен другий байт більшості рядків у Java - нульовий (Unicode поки що використовується рідко). Те, що рядки в Java вимагають удвічі більше пам'яті, ніж звичайні ASCII, не дуже лякає, поки вам для ефективної роботи з текстом в редакторах та інших подібних додатках не доведеться працювати з величезним масивом типу char.

1.5 Введення-виведення інформації з консолі

Ви вже знаєте, що інформацію можна вивести на стандартний пристрій виводу (тобто консольне вікно), викликавши метод System.out.println();
Після появи JDK 5.0 для того, щоб організувати читання інформації з консолі, вам потрібно створити об'єкт Scanner і зв'язати його зі стандартним вхідним потоком System. in.
Scanner in = new Scanner(System.in);
Зробивши це, ви отримаєте численні методи класу Scanner, призначені для читання вхідних даних. Наприклад, метод nextLine() забезпечує прийом рядка тексту.
System.out.print("KaK вас звуть?");
String name = in.nextLine();
В даному випадку ми використовували метод nextLine(), тому що вхідний рядок може містити пробіли. Для того, щоб прочитати одне слово (розділювачами між словами вважаються прогалини), можна використовувати наступний виклик:
String firstName = in.next();
Для читання цілого значення призначений метод nextInt().
System.out.print("Скільки вам років?");
int age = in.nextInt();
Як неважко здогадатися, метод nextDouble() читає чергове число у форматі з плаваючою точкою.

Програма, код якої представлений нижче в лістингу, запитує ім'я користувача та його вік, а потім виводить повідомлення типу “Вам у наступному році вам буде 46”. У першому рядку міститься вираз import java.util.*; Клас Scanner належить пакету java.util package. Якщо ви збираєтеся використовувати в програмі клас, що не міститься в базовому пакеті java.lang, вам потрібно включити директиву import до складу коду.

Приклад введення-виведення з консолі

```
import java.util.Scanner;
public class JavaIOConsole {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        // Отримання першого рядка вхідних даних
        System.out.print("Як вас звати?");
        String name = in.nextLine();
        // Отримання цілочисленного значення
        System.out.print("Скільки вам років?");
        int age = in.nextInt();
        // Відображення інформації у консольному вікні
        System.out.println(name + ", наступного року вам буде "+(age + 1));
    }
}
```

короткий опис класу scaner java

`Scanner`(З пакета java.util) - це простий текстовий сканер, який використовується для читання даних з різних джерел, таких як консоль (System.in), файли або рядки.

Основні фішки:

- **Парсинг типів: Дозволяє одразу отримувати дані потрібного типу за допомогою методів nextInt(), nextDouble(), nextLine() та ін.**
- **Розділювачі: За промовчанням використовує пробіли для поділу даних (токенів), але це можна налаштувати.**
- **Перевірка: Методи типу hasNextInt() дозволяють заздалегідь дізнатися, чи можна вважати дані конкретного типу, щоб уникнути помилок.**

Приклад:

```
Scanner sc = новий Scanner(System.in);
int num = sc.nextInt(); // Читає ціле число з консолі
```

==========