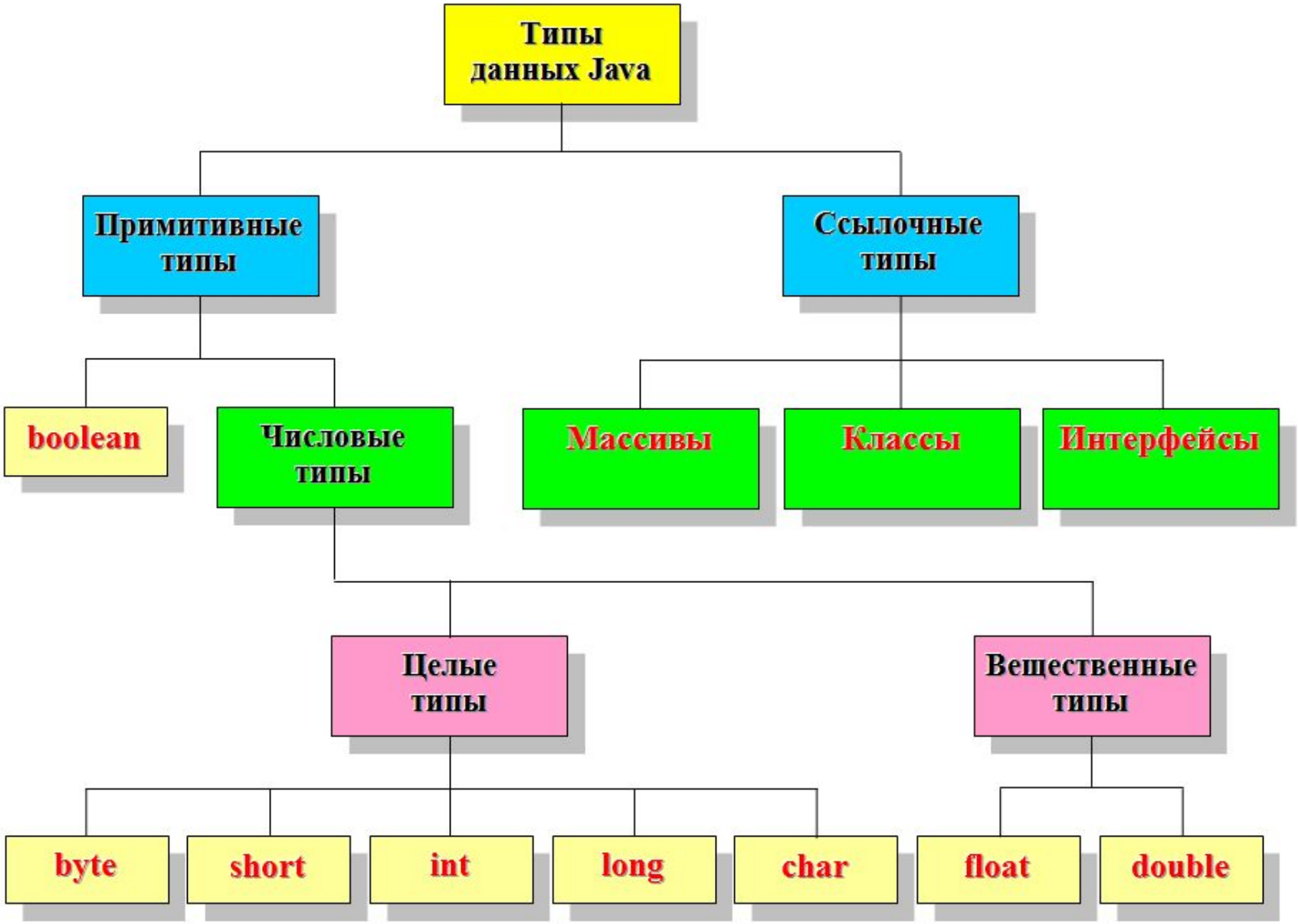
Типи даних і оголошення змінних

Язык программирования Java является **языком со строгой типизацией** (strongly typed language) - каждая переменная и каждое выражение имеют тип, который должен быть известен во время компиляции программы.

Тип ограничивает набор значений, которые могут быть присвоены переменной, либо получены в выражении, ограничивает операции над значениями и определяет реализацию конкретной операции.

В Java определено две категории данных:

- примитивные типы (primitive types);
- ссылочные типы (reference types).


Типы данных Java
Примитивные типы
Ссылочные типы
boolean
Числовые типы
Массивы
Классы
Интерфейсы
Целые типы
Вещественные типы
byte
short
int
long
char
float
double

Объявление переменной:

```
имя-типа идентификатор-переменной;
```

```
int x;
String string1;
```

Объявление нескольких переменных одного типа**:**

```
имя-типа идентификатор-переменной-1,
идентификатор-переменной-2,…;
```

```
int x1, x2, x3;
```

Переменная является указанием места хранения значения переменной в памяти. Переменная примитивного типа всегда содержит значение переменной указанного типа, а переменная ссылочного типа хранит ссылку (адрес) объекта указанного типа.

Создавать новые переменные можно в любом месте программы. Любое объявление переменной имеет свою **область действия**, границы которой зависят от того, где именно расположено это объявление.

При помещении фрагмента текста программы в пару фигурных скобок { } создается новый **блок**. Блоки могут быть вложенными. Переменная доступна в блоке только в том случае, если она определена в этом блоке или в одном из вышестоящих блоков, в который входит текущий блок.

```
{
int x;
…
      {
     int y;
     …
     }
}
```

- начало блока 1

- начало блока 2

- конец блока 2

- конец блока 1

Переменная **x** определена и в блоке 1 и в блоке 2, а переменная **y** – только в блоке **2**.

Во внутреннем блоке нельзя объявлять переменную с таким же именем, как во внешней области действия

Каждая переменная имеет **время жизни**. Когда текущий блок, в котором переменная была объявлена, заканчивается, она становится доступной для уничтожения с помощью так называемого **сборщика мусора**.

Изменить область действия и время жизни переменной можно с помощью **модификаторов доступа**. Модификаторы вставляются в объявление переменной перед *именем-типа*.

По умолчанию (без модификатора) переменная доступна только классам в том же пакете, что и класс, в котором она содержится.

Модификатор **public** определяет, что переменная доступна как внутри, так и вне класса, т. е. переменная является глобальной и доступна любому другому объекту.

Модификатор **private** означает, что переменная доступна только в том классе, в котором она была объявлена.

Модификатор **final** определяет, что переменная имеет постоянное (неизменное) значение и не может быть переопределена.

Примітивні типи даних

Булевий (логічний) тип

Величины типа **boolean** принимают значения **true** или **false**.

Цілочисельні типи

Тип	Число байтов	Диапазон значений	Описание
byte	1	-128..127	Однобайтовое целое число (8-битовое целое со знаком)
short	2	$-2^{15}..2^{15}-1$ = -32768.. 32767	Короткое целое число (16-битовое целое со знаком)
char	2	\u0000..\uFFFF=0.. 65535	Символьный тип (беззнаковое 16-битовое целое)
int	4	$-2^{31}..2^{31}-1$ = $-2.147483648 \cdot 10^{9}..$ $2.147483647 \cdot 10^{9}$	Целое число (32-битовое целое со знаком)
long	8	$-2^{63}..2^{63}-1$ = $-9.22337203685478 \cdot 10^{18}..$ $9.22337203685478 \cdot 10^{18}$	Длинное целое число (64-битовое целое со знаком)

Примеры объявления целых переменных:

```
int i,j,k;
int j1;
byte i1, i2=-5;
short i3=-15600;
long m1=1, m2, m3=-100;
```

```
byte a1 = 0xF1, a2 = 0x07;
short r1 = 017;
```

При объявлении в классе значение целочисленной переменной по умолчанию равно нулю

В методе все переменные перед использованием обязательно нужно инициализировать

Константами называются именованные ячейки памяти с неизменяемым содержимым. Объявление констант осуществляется в классе, при этом перед именем типа константы ставится комбинация зарезервированных слов **public** и **final**:

```
public final int MAX1=255;
public final int MILLENIUM=1000;
```

Робота з числами в мові Java

Для хранения отрицательных чисел используется ***дополнительный код***. Машинное представление положительного числа n переводится в машинное представление числа $n_2 = -n$ по следующему алгоритму:

1. Число n преобразуется в число n_1 путем замены всех единиц числа n нулями, а нулей единицами.

2. Перевод n_1 в $n_2 = n_1 + 1$ (т. е. к получившемуся числу n_1 добавляется единица младшего разряда).

Основні оператори для роботи з цілочисельними величинами

+ Оператор сложения

– Оператор вычитания

***** Оператор умножения

/ Оператор деления

% Оператор остатка от целочисленного деления

= Оператор присваивания

++ Оператор инкремента (увеличения на 1)

+= **v+=i** эквивалентно **v=v+i**

-= **v-=i** эквивалентно **v=v-i**

***=** **v*=i** эквивалентно **v=v*i**

/= **v/=i** эквивалентно **v=v/i**

%= **v%=i** эквивалентно **v=v%i**

Integer.parseInt(строка)
Long.parseLong(строка)

- преобразование строкового представления числа в целое значение

Побітові операції

Побитовые операции рассматривают числовые значения как поля битов

Сдвиг влево с учетом знака <<

```
int x =  31, z;     // x = 00000000 00000000 00000000 00011111
z = x << 2;  // z = 124: 00000000 00000000 00000000 01111100
```

Сдвиг вправо с учетом знака >>

```
int x =  -17, z;   // x: = 11111111 11111111 11111111 11101111
z = x >> 2;  // z = -5: 11111111 11111111 11111111 11111011
```

Сдвиг вправо без учета знака >>>

```
int x =  -17, z;   // x = 11111111 11111111 11111111 11101111
z = x >>> 2;       // z = 1073741819
          // z:   00111111 11111111 11111111 11111011
```

Побитовое И &

```
int x =  112;      // x = 00000000 00000000 00000000 01110000
int y = 94;   // y = 00000000 00000000 00000000 01011110
int z;
z = x & y;    // z=80: 00000000 00000000 00000000 01010000
```

Побитовое ИЛИ |

```
int x =  112;      // x = 00000000 00000000 00000000 01110000
int y = 94;   // y = 00000000 00000000 00000000 01011110
int z;
z =x | y;      // z = 126: 00000000 00000000 00000000 01111110
```

Побитовое исключающее ИЛИ ^

```
int x =  112;      // x = 00000000 00000000 00000000 01110000
int y = 94;   // y = 00000000 00000000 00000000 01011110
int z;
z =x ^ y;      // z = 46: 00000000 00000000 00000000 00101110
```

Операції порівняння

Эти операции имеют два операнда и возвращают булево значение, соответствующее результату сравнения (**false** или **true).**

"==" (равно), "!=" (не равно),
">" (больше), ">=" (больше или равно),
"<" (меньше) "<=" (меньше или равно)

Пример:

```
boolean isEqual, isNonEqual, isGreater,
isGreaterOrEqual, isLess, isLessOrEqual;
int x1 = 5, x2 = 5, x3 = 3, x4 = 7;
  isEqual = x1 == x2;                  // isEqual = true
  isNonEqual = x1 != x2;               // isNonEqual = false
  isGreater = x1 > x3;                 // isGreater = true
  isGreaterOrEqual = x2 >= x3;              // isGreaterOrEqual = true
isLess = x3 < x1;                   // isLess = true
isLessOrEqual = x1 <= x3;
```

Булеві операції

Выполняются над **boolean** переменными и их результатом также является значение типа **boolean**.

отрицание "!"

И "&"

исключающее ИЛИ "^"

ИЛИ "|"

Кроме того, к **boolean** операндам применимы операции **"==" (равно)** и **"!=" (не равно).**

Укороченные (short-circuit) логические операции

укороченное И "&&"

укороченное ИЛИ "||"

При использовании этих операций второй операнд вообще не будет вычисляться, что полезно в тех случаях, когда правильное функционирование правого операнда зависит от того, имеет ли левый операнд значение **true** или **false**.

Примеры булевых операций:

```
boolean isInRange, isValid, isNotValid,
isEqual, isNotEqual;
int x = 8;
isInRange = x > 0 && x < 5;          // isInRange = false
isValid = x > 0 || x > 5;            // isValid = true
isNotValid = !isValid;               // isNotValid = false
isEqual = isInRange == isValid;      // isEqual = false
isNotEqual = isInRange != isValid  // isNotEqual = true
```

Пріоритет операторів

Приоритет	Группа операторов	Операторы				
1 *высший*	Постфиксные	( )	[ ]	.		
2	Унарные	++ операнд операнд ++	--операнд операнд--	~	!	+ операнд - операнд
3	Создания объектов и преобразования типа	new	(тип) операнд			
4	Мультипликативные	*	/	%		
5	Аддитивные	+	-			
6	Сдвиги битов	>>	>>>	<<		
7	Отношения	>	>=	<	<=	instanceof
8	Эквивалентности	==	!=			
9	Побитовое И	&				
10	Побитовое исключающее ИЛИ	^				
11	Побитовое ИЛИ	|				
12	Логическое И	&&				
13	Логическое ИЛИ	||				
14	Условный	? :				
15 *низший*	Присваивания	=	Оператор = (+=, -=, *=, /= и т. п.)			

Символьний тип char

Символы в Java определяются с помощью ключевого слова **char** и реализованы с использованием стандарта Unicode. Можно задать константу-символ в программе как обычный символ. Символьное значение должны быть заключено в пару одиночных апострофов.

```
char symbol='f';
```

Другой способ записи символов: пара символов "**\u**", за которой следует четырехзначное шестнадцатеричное число (в диапазоне от 0000 до FFFF), представляющее собой код символа **в** Unicode.

```
char symbol = '\u0042';
```

Также символьной переменной можно присваивать числовой код символа Unicode (номер символа в кодовой таблице)

```
char c1='a';
char c2='\u0061';
char c3=97;
```

Дійсні типи

Тип	Размер	Диапазон	Описание
`float`	4 байта (32 бита)	$1.5 \cdot 10^{-45}..3.4 \cdot 10^{38}$	Одинарная точность, 7—8 значащих десятичных цифр мантиссы. Тип `real*4` стандарта IEEE754
`double`	8 байт (64 бита)	$5 \cdot 10^{-324}..1.7 \cdot 10^{308}$	Двойная точность, 15—16 значащих цифр мантиссы. Тип `real*8` стандарта IEEE754

Для чисел с плавающей точкой нужно указывает целую и дробную часть, разделенные точкой (4.6, 7.0 и т.п.).

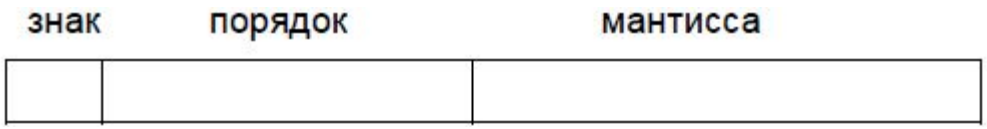
Для больших чисел можно использовать экспоненциальную форму записи (для отделения мантиссы от порядка используется символ "e" или символ "E"), например, число $-3{,}58 \cdot 10^{7}$ записывается как –3.58E7, а число $73{,}675 \cdot 10^{-15}$ – как 73.675e-15.

Двоичное представление вещественных чисел

Число x с плавающей точкой можно представить в виде:

$$x = s \cdot m \cdot 2^p.$$

s — знак числа,
m — мантисса,
p — порядок числа.



Если знаковый бит равен 0, то число положительное, если 1 - отрицательное.

Набор битов, хранящийся в мантиссе, задает положительное число ***m***, лежащее в пределах $\mathbf{1 \le m < 2}$.

При перемножении чисел их мантиссы перемножаются, а порядки складываются. При делении мантиссы делятся, а порядки вычитаются. И умножение, и деление мантисс происходит по тем же алгоритмам, что и для целых чисел. Но при выходе за размеры ячейки отбрасываются не старшие, а младшие байты.

Переменные с плавающей точкой могут хранить не только численные значения, но и любой из особо определенных флагов (состояний):

- отрицательная бесконечность;
- отрицательный нуль;
- положительная бесконечность;
- положительный нуль;
- «отсутствие числа» (not-a-number, **NaN).**

Операторы для работы с вещественными величинами аналогичны операторам для работы с целочисленными величинами

В классе java.lang.Math заданы константы π = 3,14... (Math.PI) и e = 2,71... (Math.E), а также математические функции.

Функції, задані в класі Math

Функция	Примечание
Тригонометрические и обратные тригонометрические функции	
`sin(x)`	sin(*x*) — синус
`cos(x)`	cos(*x*) — косинус
`tan(x)`	tg(*x*) — тангенс
`asin(x)`	arcsin(*x*) — арксинус
`acos(x)`	arccos(*x*) — арккосинус
`atan(x)`	arctg(*x*) — арктангенс
`toRadians(angdeg)`	angdeg / 180.0 · *PI*; — перевод углов из градусов в радианы
`toDegrees(angrad)`	angrad · 180.0 / *PI*; — перевод углов из радиан в градусы
Случайное число, остаток	
`random()`	Псевдослучайное число в диапазоне от 0,0 до 1,0. При этом: `0 <= Math.random() < 1`
`IEEEremainder(x,y)`	Погрешность целочисленного деления *x* на *y*, т. е. разность: *x* — *y* * *n*, где *n* — результат целочисленного деления

Функції, задані в класі Math

Степени, экспоненты, логарифмы	
`exp(x)`	e^x — экспонента
`expm1(x)`	$e^x - 1$. При x, близком к 0, дает гораздо более точные значения, чем $\exp(x) - 1$
`log(x)`	$\ln(x)$ — натуральный логарифм
`log10(x)`	$\log_{10}(x)$ — десятичный логарифм
`log1p(x)`	$\ln(1 + x)$. При x, близком к 0, дает гораздо более точные значения, чем $\log(1 + x)$
`sqrt(x)`	$\sqrt{x}$ — квадратный корень
`cbrt(x)`	$\sqrt[3]{x}$ — кубический корень
`hypot(x,y)`	$\sqrt{x^2 + y^2}$ — вычисление длины гипотенузы по двум катетам
`pow(x, y)`	x^y — возведение x в степень y
`sinh(x)`	$\mathrm{sh}(x) = \frac{e^x - e^{-x}}{2}$ — гиперболический синус
`cosh(x)`	$\mathrm{ch}(x) = \frac{e^x + e^{-x}}{2}$ — гиперболический косинус
`tanh(x)`	$\mathrm{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ — гиперболический тангенс

Функції, задані в класі Math

Модуль, знак, минимальное, максимальное число	
`abs(m), abs(x)`	Абсолютное значение числа. Аргумент типа `int`, `long`, `float` или `double`. Результат того же типа, что аргумент
`signum(a), signum(x)`	Знак числа. Аргумент типа `float` или `double`. Результат того же типа, что аргумент
`min(m,n), min(x,y)`	Минимальное из двух чисел. Аргументы одного типа. Возможны типы: `int`, `long`, `float`, `double`. Результат того же типа, что аргумент
`max(m,n), max(x,y)`	Максимальное из двух чисел. Аргументы одного типа. Возможны типы: `int`, `long`, `float`, `double`. Результат того же типа, что аргумент
Округления	
`ceil(x)`	Ближайшее к *x* целое, большее или равное *x*
`floor(x)`	Ближайшее к *x* целое, меньшее или равное *x*
`round(a)` `round(x)`	Ближайшее к *x* целое. Аргумент типа `float` или `double`. Результат типа `long`, если аргумент `double`, и типа `int` — если `float`. То же, что `(int)floor(x + 0.5)`
`rint(x)`	Ближайшее к *x* целое
`ulp(a), ulp(x)`	Расстояние до ближайшего большего чем аргумент значения того же типа (дискретность изменения чисел в формате с плавающей точкой вблизи данного значения). Аргумент типа `float` или `double`. Результат того же типа, что аргумент

Приведення типів при виконанні операцій

```
double y;
byte x;
y = x + 5;
```

При выполнении операции присваивания преобразование типов происходит автоматически, если происходит **расширяющее преобразование** (widening conversion) и **два типа совместимы**.

byte→short→int→long→float→double

Числовые типы не совместимы с типами char и boolean.

Если необходимо выполнить **сужающее преобразование**, используется операция **приведения** (***cast***) типа, которая имеет следующий формат:

```
(тип-преобразования) значение
```

```
byte x = 71;
char symbol = (char) x;
```

Если значение с плавающей точкой присваивается целому типу, то при явном преобразовании типа происходит также усечение (truncation) числа.

При выполнении арифметических и побитовых преобразований все значения **byte** и **short**, а также **char** расширяются до **int**, затем, если хотя бы один операнд имеет тип **long**, тип целого выражения расширяется до **long**. Если один из операндов имеет тип **float**, то тип полного выражения расширяется до **float**, а если один из операндов имеет тип **double**, то тип результата будет **double**.

Автоматические расширения типов (особенно расширения short и byte до int) могут вызывать плохо распознаваемые ошибки во время компиляции.

```
byte x = 30, y =5;
x = x + y;
```

Чтобы этого избежать надо использовать во втором операторе явное преобразование типов

```
byte x = 30, y =5;
x = (byte) (x + y);
```

Масиви в Java

Массивом называется именованный набор переменных одинакового типа. Каждый элемент массива является переменной, которая однозначно определяется путем указания **имени** массива и целочисленной позиции элемента в массиве – **индекса** элемента в массиве.

Массивы в Java являются гибридом объектов и примитивных типов.

Описание массива производится в три этапа

На **первом** этапе выполняется **объявление массива** (array declaration).

```
имя-типа [ ] идентификатор-массива
```

либо

```
имя-типа  идентификатор-массива[ ]
```

имя-типа – имя примитивного или ссылочного типа
идентификатор-массива – имя, присваиваемое массиву.

```
int [ ] myArray;
```

```
int myArray[ ];
```

Если несколько массивов имеют одинаковый тип, их также можно объявить в списке, используя первый формат записи

```
short [ ] a, b;
```

На **втором** этапе, этапе **определения** или **создания массива** (array instantiation) указывается количество элементов массива, называемое его длиной, выделяется место для массива в оперативной памяти, переменная-ссылка получает адрес массива.

```
new имя-типа [размер-массива]
```

размер-массива – переменная или выражение любого целочисленного типа, за исключением типа **long**

```
myArray = new int[5];
byte dim1 = 10, dim2 = 2;
a = new short[dim1];
b = new short[dim1 + dim2];
```

На **третьем** этапе производится **инициализация массива** (array initialization). Элементы массивов числовых типов получают нулевое значение соответствующего типа, элементы булевских массивов – значение **false**, а элементы массивов ссылочных типов – значения **null**.

Массивы могут быть инициализированы во время создания. При этом ***размер-массива*** в квадратных скобках не указывается, а после закрывающей квадратной скобки в фигурных скобках задается список значений элементов массива, разделенных запятыми.

```
myArray = new int[ ]{0, 1, 2, 3, 4};
```

Можно объединить первый и второй этапы в одном операторе

```
int [ ] myArray = new int[5];
```

Поскольку при инициализации массива размер массива указывать не надо, можно объединить первый и третий этап в одном операторе

```
int [ ] myArray = {0, 1, 2, 3, 4};
```

Доступ до елементів масиву

Элемент массива, помимо значения, характеризуется своей позицией в массиве – **индексом**. Индексом элемента является значение типа **byte**, **short**, **int** или **char**, заключенное в квадратные скобки, например **myArray[4]**.

Индекс элемента массива изменяется от нуля до величины, на единицу меньшей размера массива

```
int myArrayLength = myArray.length;
```

Багатовимірні масиви

У массивов в Java должна быть указана, по крайней мере, одна размерность, однако в программах могут использоваться и многомерные массивы, причем остальные размерности можно определять во время выполнения программы.

Элементами массивов в Java могут быть другие массивы.

```
int [ ][ ] matrix = new int[3][4];
```

Первый (левый) размер массива должен задаваться при его создании. Другие размеры могут указываться позже.

Таким образом, можно создавать непрямоугольные массивы

Пример

```
int [ ] [ ] triangleMatrix = new int [3] [ ] ;
triangleMatrix[0] = new int [1];
triangleMatrix[0][0] = 1;
triangleMatrix[1] = new int [2];
triangleMatrix[1][0] = 2;
triangleMatrix[1][1] = 3;
triangleMatrix[2] = new int [3];
triangleMatrix[2][0] = 4;
triangleMatrix[2][1] = 5;
triangleMatrix[2][2] = 6;
```

Матрица **triangleMatrix** будет треугольной, и будет иметь следующий вид:

1
2 3
4 5 6

Змінні перечислимого типу

Определяются с помощью ключевого слова **enum.**

Объявление перечислимой переменной

```
модификаторы-класса enum идентификатор-класса {
        имя-константы-1, …, имя-константы-n
}
```

список имен констант задает значения, которые может принимать переменная ***идентификатор-класса***

Пример

```
public enum Month {
JANUARY, FEBRUARY, MARCH, APRIL,MAY, JUNE, JULY,
AUGUST, SEPTEMBER, OCTOBER, NOVEMBER, DECEMBER
}
```

```
Month currentMonth;
currentMonth = Month. SEPTEMBER;
```

Объявление перечислимой переменной является объявлением класса и может иметь более сложный вид

модификаторы-класса **enum** *идентификатор-класса* **{**
имя-константы-1(***список-аргументов***), …,
имя-константы-n(***список-аргументов***);
тело-класса
}

Список-аргументов задает отделяемые друг от друга запятыми аргументы, которые передаются конструктору класса при создании объекта данного класса. В теле класса, помимо конструктора, могут быть определены и другие методы для работы с переменными объекта.

Пример

```
public enum Month {
JANUARY(1,31), FEBRUARY(2,28), MARCH(3,31), APRIL(4,30),
MAY(5,31), JUNE(6,30), JULY(7,31), AUGUST(8,31), SEPTEMBER(9,30),
OCTOBER(10,31), NOVEMBER(11,30), DECEMBER(12,31);
      private int monthIndex;
      private int numberOfDaysInMonth;
      Month(int monthIndex, int numberOfDaysInMonth){
            this.monthIndex = monthIndex;
            this.numberOfDaysInMonth = numberOfDaysInMonth;
      }
      int getMonthIndex()
      {
            return monthIndex;
      }
int getNumberOfDaysInMonth() {
            return numberOfDaysInMonth;
      }
}
```

Использование переменной в другом классе:

```
Month currentMonth;
currentMonth = Month. SEPTEMBER;
int currentMonthIndex = currentMonth.getMonthIndex();
int currentNumberOfDaysInMonth =
currentMonth.getNumberOfDaysInMonth();
```

В результате выполнения этого фрагмента программы переменной **currentMonthIndex** будет присвоено значение **9,** а переменной **currentNumberOfDaysInMonth –** значение **30.**

Оператори управління Java

Умовний вираз...?... : ...

```
условие?значение1:значение2
```

Когда ***условие*** имеет значение **true**, функция возвращает ***значение1***, в противном случае — ***значение2***.

В выражениях для условий и значений можно использовать пробелы, в том числе отделять ими символы **?** и **:**.

```
j=i<5?i+1:i+2
```

или

```
j=(i<5)?(i+1):(i+2)
```

Складений оператор

Согласно синтаксису языка Java во многих конструкциях возможен только один оператор, но часто встречается ситуация, когда необходима последовательность из нескольких операторов.
Тогда используется составной оператор - блок кода между фигурными скобками {}.

```
оператор
{
    последовательность операторов
}
```

или

```
оператор{
    последовательность операторов
}
```

Второй способ установлен по умолчанию в среде NetBeans.

Оператори передачі управління в Java

При работе программы операторы выполняются последовательно. Однако часто бывает необходимым изменить порядок выполнения операторов. Для этого используются операторы управления программой.

- **условный оператор;**
- **операторы цикла;**
- **операторы перехода;**
- **оператор выбора.**

Умовний оператор

```
if (булево-выражение)
операторы-1;
else
операторы-2;
```

Если значение ***булевого-выражения*** равно **true**, то выполняются ***операторы-1***, иначе выполнятся ***операторы-2***. Оператор **else** может быть опущен, в этом случае, если ***булево-выражение*** равно **false**, выполняется оператор, следующий за оператором **if**.
Если ***операторы-1*** или ***операторы-2*** содержат несколько операторов, они должны быть заключены в фигурные скобки.

```
if (x == 0)
      y = 0;
else {
      y = 1;
      z = x + 1;
}
```

Пример 1

```
if(a<b)
      a=a+1;
else
      if(a==b)
            a=a+1;
      else {
            a=a+1;
            b=b+1;
      }
```

Пример 2

```
if(условие)
      оператор1;
      оператор2;
else
      оператор3;
```

```
if(условие) {
      оператор1;
      оператор2;
} else
оператор3;
```

Пример 3

```
if(условие)
      оператор1;
else
      оператор2;
оператор3;
```

```
if(условие)
      оператор1;
else {
      оператор2;
      оператор3;
}
```

Оператори циклу

- **оператор for;**
- **расширенный оператор for;**
- **оператор while;**
- **оператор do…while.**

Оператор цикла for

for (*инициализация-цикла*;*контрольное-выражение*; *шаговое-выражение*)
тело-цикла

Инициализация-цикла выполняется один раз перед первым проходом тела цикла

Контрольное-выражение вычисляется перед началом каждого выполнения тела цикла

Шаговое-выражение вычисляется в конце каждого выполнения тела цикла

Тело цикла может содержать один оператор или несколько операторов, заключенных в фигурные скобки.

```
int a[ ] = {1, 2, 3, 4, 5};
int sum = 0;
for (int i = 0; i < a.length; i++)
sum+=a[i];
```

В выражение для инициализации цикла и в выражение для итерации цикла можно включить **несколько операторов, разделенных запятыми**. Обычно такая форма применяется, когда в цикле необходимо использовать несколько переменных цикла

```
boolean isSymmetric = true;
…
for(int i=0, j=a.length-1; i < j; i++, j--)
if(a[i] != a[j])
isSymmetric = false;
```

Расширенный оператор for

for(*модификаторы-переменной* *тип идентификатор-переменной* **:** *выражение***)**
тело-цикла

Выражение в этом цикле должно задавать набор значений переменной. Если набор значений является массивом, в качестве выражения задается имя массива. Набор элементов переменной перечислимого типа и коллекций можно получить с помощью статического метода **values().** В процессе выполнения цикла переменной с именем ***идентификатор-переменной*** будут последовательно присваиваться значения всех элементов массива, переменной перечислимого типа или коллекции.

```
int a[ ] = {1, 2, 3, 4, 5};
int sum = 0;
for (int ai : a)
sum+=ai;
```

```
int dayNumber = 0;
for (Month monthN : Month.values())
dayNumber+=
monthN.getNumberOfDaysInMonth();
```

Операторы цикла while и do…while

while(*булево-выражение*)
тело-цикла

do
тело-цикла
while(*булево-выражение*);

Оба цикла выполняются до тех пор, пока ***булево-выражение*** имеет значение **true,** однако в цикле **while** вычисление ***булевого-выражения*** производится до начала очередного выполнения цикла, а в цикле **do…while** – после его очередного выполнения.

```
int a[ ] = {1, 2, 3, 4, 5};
int sum = 0, i = 0;
while (i < 0)
{
      sum+=a[i];
      i ++;
}
```

```
int a[ ] = {1, 2, 3, 4, 5};
int sum = 0, i = 0;
do
{
      sum+=a[i];
      i ++;
}
while (i < 0);
```

Оператори переходу

В Java отсутствует оператор **goto** *метка*

Оператор прерывания

break *метка*;

передает управление за пределы цикла или оператора выбора, помеченного указанной ***меткой***. ***Метка*** представляет собой обычный идентификатор Java, за которым следует двоеточие. ***Метка*** может быть опущена – в этом случае управление передается за пределы цикла или оператора выбора, содержащего данный оператор **break.**

```
int a[ ] = {1, 2, 3, 0, 5};
int i, zeroIndex = -1;
for (i = 0; i < a.length; i++)
if(a[i] == 0)
break;
if (i < a.length)
zeroIndex = i;
```

- использование оператора break без метки

```
int a[ ][ ] =
{{1,2,3},{5,6,7},{8,9,0},{11,12,13}};
int i, j;
…
iLoop:
for(i = 0; i < 4; i++)
{
      for(j = 0; j < 3; j++)
      {
            if (a[i][j] == 0)
                  break iLoop;
      }
}
```

- использование оператора break с меткой

Оператор продолжения

continue *метка*;

Передает управление внутри цикла.

Если метка опущена, оператор **continue** передает управление на самый конец тела цикла и, если контрольное выражение, вычисленное после того, как очередной проход тела цикла был завершен оператором **continue**, будет равно **true,** выполнение цикла продолжится**.**

Это бывает полезно для того, чтобы пропустить при выполнении тела цикла некоторые из операторов. Если метка указана, операторы пропускаются в цикле, помеченном указанной меткой.

```
int a[ ] = {1, 2, 3, 4, 5};
int sum = 0;
for (int i = 0; i < a.length; i++)
{
if(a[i]%2 != 0)
continue;
sum += a[i];
}
```

Оператор вибору

Оператор выбора **switch** обычно используется, когда необходимо организовать ветвление программы по нескольким направлениям, однако условия проверки в нем должны быть выражены с помощью различных значений целой переменной.

```
switch (выражение)
{
case значение-1: операторы-1;
case значение-2: операторы-2;
…
default: операторы-n;
}
```

Выражение, которое ставится в круглых скобках после ключевого слова **switch**, может принадлежать к одному из типов **char, byte, short** или **int**, а также одним из значений переменной перечислимого типа**.**

Проверка значения ***выражения*** на равенство заданному константному ***значению*** осуществляется с помощью операторов **case,** входящих в оператор **switch.**
Все операторы **case** должны содержать различные значения.
Как только один из операторов **case** опознал значение ***выражения***, управление получает операторы, следующие после символа **":"** за этим оператором **case.**

После этого выполняются операторы, содержащиеся во всех следующих операторах **case** и в операторе **default.** Для того, чтобы обойти выполнение последующих операторов **case** и оператора **default,** следует задать последним оператором для данного **case** оператор **break.** Если значение ***выражения*** не совпадает ни с одним из ***значений***, выполняются операторы, следующие за оператором **default** (этот оператор может быть опущен).

Операторы, следующие за двоеточием, можно не заключать в фигурные скобки.

Если для нескольких значений выполняются одни и те же действия, операторы **case** можно объединить.

```
case значение-1:
case значение-2: операторы;
```

```
char s;
int x;
switch(s) {
case 'a':
      x = 0;
      break;
case 'A':
      x = 1;
      break;
case ' ':
case '(':
case ')':
      x = 2;
      break;
default:
      x = 3;
}
```