## II. ОСНОВИ МОВИ *JAVA*

### 2.1. Типи даних, змінні, масиви

*Java* – мова, яка чітко дотримується типів, що значною мірою гарантує безпеку програм і стійкість до помилок. Що це означає? По-перше, кожна змінна має свій тип, і кожний тип чітко визначено. По-друге, усі присвоєння, які виконують в явному вигляді або через пересиляння параметрів у методи, перевіряються на співпадання типів. У *Java* не передбачено автоматичне приведення типів або перетворення конфліктуючих типів, як в інших мовах. Компілятор *Java* перевіряє усі вирази і параметри щодо сумісності типів.

#### 2.1.1. Прості типи

До простих типів у *Java* належать чотири групи:

1. Цілочисельні: *byte, short, int* і *long*, які призначені для цілих зі знаком;
2. Числа з плаваючою комою: *float* і *double* – числа з дробовою частиною;
3. Символи: *char* − всі символи, включаючи букви і цифри;
4. Логічні: *boolean* – "істина" і "фальш".

Ці типи використовують у програмах, об’єднують в масиви тощо.

Чому введено термін *прості типи*? Тому, що вони не є об’єктами. Незважаючи на те, що в усьому іншому *Java* – цілковито об’єктно-орієнтована мова, прості типи є винятком. Все пояснюється ефективністю. Перетворення простих типів в об’єкти спричинило б до суттєвого зниження продуктивності.

Для простих типів мови *Java* явно задається діапазон значень і чітко визначені правила виконання операцій над ними. У мові *С++* діапазон цілочисельних значень може залежати від середовища виконання програми. У мові *Java*, виходячи з потреб мобільності, усі типи даних мають строго визначений діапазон. Це дає змогу писати програми з впевненістю, що вони будуть працювати на комп’ютерах з будь-якою архітектурою. Строге задання розмірів цілого приводить до деякого зниження швидкості виконання програм у деяких середовищах, однак цього не уникнути, коли головною метою є мобільність.

***Цілочисельні типи.*** У *Java* визначено чотири цілочисельні типи: *byte, short, int* і *long*. Усі вони визначають знакові, тобто додатні і від’ємні значення. *Java* не підтримує беззнакові цілі. У табл. 2.1 наведено ширину і діапазони цілочисельних типів.

Т а б л и ц я 2.1. **Ширина в бітах і діапазони значень цілочисельних типів**

|  |  |  |
| --- | --- | --- |
| Тип | Ширина | Діапазон |
| *long* | 64 | від - 9223372036854775808 до 9223372036854775807 |
| *int* | 32 | від – 2147483648 до 2147483647 |
| *short* | 16 | від – 32768 до 32767 |
| *byte* | 8 | від – 128 до 127 |

Ширину цілочисельного типу, не дивлячись на те, що її вимірюють в бітах, треба розглядати не як об’єм пам’яті, яку він займає, а як поведінку, визначену для змінних і виразів заданого типу. Виконуване середовище *Java* використовує свою кількість байтів, яка краще відповідає апаратній платформі, забезпечуючи у цьому випадку поведінку, визначену через заданий програмістом тип. Скоріше всього виконуване середовище *Java* відводить на деяких платформах 32 біти для типів *byte* і *short*.

Змінні типу *byte* особливо корисні при роботі з потоком даних або файлом, а також з неформатованими даними.

*Short* – цей тип рідко використовують, тому що розміщення старшого і молодшого байтів не завжди збігається з реальним підходом, який використовують переважно на 16-тирозрядних комп’ютерах.

*Int* – найвживаніший цілочисельний тип. Його необхідно використовувати при створенні лічильника, індексації масивів, виконанні обчислень над цілими числами. Іноді здається, що використання *byte* чи *short* економить пам’ять, однак найвірогідніше, що конкретне середовище виконання все одно перетворить ці типи в *int*. Необхідно пам’ятати, що тип в мові *Java* визначає поведінку, а не розмір пам’яті. Єдиним винятком є масив, коли *byte* гарантує використання 8-ми бітів на елемент масиву, *short* – 16-ти бітів, а *int* – 32-х бітів.

Змінні типу *long* необхідні в тих випадках, коли тип *int* є замалим для збереження бажаного результату.

***Дійсні типи.*** У *Java* існує два дійсних типи *float* і *double* для представлення чисел з одинарною і подвійною точністю відповідно. Основні характеристики цих типів наведено в табл. 2.2.

Т а б л и ц я 2.2. **Ширина в бітах і діапазони значень дійсних типів**

|  |  |  |
| --- | --- | --- |
| Тип | Ширина | Діапазон |
| *double* | 64 | від –1.7е-308 до 1.7е+308 |
| *float* | 32 | від –3.4е-038 до 3.4е+038 |

Змінні типу *float* використовують для представлення чисел з дробовою частиною, проте з незначною точністю. На деяких процесорах операції зі змінними типу *float* виконуються швидше, ніж з типами *double* і вимагають менше пам’яті. Проте коли значення стають дуже великими або дуже малими втрачається точність обчислень.

Тип *double* використовують для представлення дійсних чисел з подвійною точністю і він вимагає 64 біти пам’яті.

На деяких сучасних процесорах, оптимізованих для високошвидкісних математичних обчислень, операції з подвійною точністю виконуються швидше ніж, з одинарною. Наприклад, усі математичні функції такі як *sin(), cos(), sqrt()* повертають значення типу *double*. Якщо необхідно виконати багато ітерацій або оперувати з великими числами, краще вибирати *double*.

***Символи.*** Виходячи з того, що мову *Java* створено для всесвітнього використання, для кодування символів використовують Unicode (повну інформацію можна отримати за адресою [http://www.unicode.org).](http://www.unicode.org)/) Unicode визначає повний міжнародний символьний набір, який дає змогу представляти усі символи мов усіх народів світу. Тому необхідно для символу 16 бітів. Отже, *char* у мові *Java* є 16-тибітовим типом. Значення типу *char* є цілі числа з діапазону від 0 до 65536. Стандартний набір символів, відомий як ASCII, займає інтервал від 0 до 127, а розширений 8мибітовий символьний набір (ISO-Latin-1) - від 0 до 255. Значення змінних *char* можна додавати та інкрементувати.

***Логічний тип.*** У мові *Java* є тип *boolean*, який використовується для задання двох значень: *true* ("істина") і *false* ("фальш"). На відміну від мови *С++*, де будь-яке число відмінне від нуля є істина, в мові *Java* цей тип не є числом.

#### 2.1.2. Літерали

***Цілі літерали.*** Будь-яке ціле число є цілим літералом. У *Java* можна використовувати цілі числа у десятковій, вісімковій і шістнадцятковій системі числення. Вісімкові літерали розпочинаються з нуля, тому 0 не використовують на початку десяткових чисел. Наприклад, значення 09 викличе помилку компілятора, тому що це запис цілочисельного вісімкового літералу і "9" не є цифрою вісімкової системи числення (від 0 до 7). Шістнадцятковий літерал розпочинається з символів ∅х або ∅Х, за якими знаходяться декілька шістнадцяткових цифр. Можна використовувати верхній і нижній регістри для цифр А-F.

Цілі літерали створюють значення типу *int*. Незважаючи на те, що мова *Java* чітко типізована, цілі літерали можна присвоювати усім цілочисельним типам (для *byte* і *short* літеральні значення не повинні виходити за діапазони значень). Для задання літерала типу *long* необхідно вказати про це компілятору, додаванням наприкінці букви L або *l*.

***Літерали з плаваючою комою.*** Дійсні числа, які є літералами з плаваючою комою, можуть бути задані у вигляді стандартної або наукової нотації. Стандартна нотація − це число, крапка, дробова частина (2.3, 3.1415926, 0.667). Наукова нотація − це стандартна нотація плюс суфікс, який задає степінь числа 10, на який має помножитися дійсне число (6.022Е23, 31415Е-05, 2е+100).

Дійсні літерали за замовчуванням перетворюються в *Java* до подвійної точності. Для задання літерала типу *float* необхідно додати в кінці літералу F або f. Можна також явно вказати, що це літерал з подвійною точністю, задавши наприкінці букву D або d.

***Логічні літерали.*** Існує два значення *true* i *false*, які мають тип *boolean*, і є логічними літералами. Ці значення не приводяться ні до 1, ні до 0. У мові *Java* їх можна присвоїти тільки змінним, оголошеним як *boolean*, або використати у виразах для логічних операторів.

***Символьні літерали.*** Літеральний символ подається в середині одинарних лапок: 'a', 'z', 'D' і т. д. Для символів, які не можна ввести з клавіатури, задаються еscape-послідовності. У табл.2.3 наведено деякі приклади символьних літералів.

Т а б л и ц я 2.3. Задання символьних літералів

|  |  |  |
| --- | --- | --- |
| Описання або еscapeпослідовність | Послідовність | Результат |
| Будь-який символ, напрклад, у | 'y' | y |
| Повернення на один символ (BS) | '\b' | повернення (backspace) |
| Горизонтальна табуляція (HT) | '\t' | табуляція (tab) |
| Перевід стрічки (LF) | '\n' | переведення рядка (новий рядок) |
| Перевід сторінки (FF) | '\f' | переведення сторінки (нова сторінка) |
| Повернення каретки (CR) | '\r' | повернення каретки |
| Подвійна лапка | \" | " |
| Одинарна лапка | \' | ' |
| Обернена риска | \\ | \ |
| Вісімковий набір бітів | \ddd | вісімковий символ ddd |
| Шістнадцятковий набір бітів | \хdd | шістнадцятковий символ dd |
| Символ Unicode | \udddd | символ з кодом dddd в  Unicode |

***Рядкові літерали.*** Рядкові літерали задають шляхом обрамлення подвійними лапками послідовності символів: "Добрий день", "дві\nстрічки", "\"Фраза у подвійних лапках\"".

Важливо пам’ятати, що рядкові літерали повинні починатися і закінчуватися в одному рядку, оскільки в мові *Java* немає еscapeпослідовності продовження рядка. Зауважимо, що рядки в *Java* є об'єктами класу *Strіng*, а не масивами символів як *С++*.

#### 2.1.3. Змінні

Змінна - це базова одиниця для збереження інформації в програмі мовою Java. Вона визначається комбінацією типу, ідентифікатора і необов’язкової ініціалізації. Змінна має область видимості і час існування.

***Оголошення змінної.*** У мові *Java* всі змінні повинні бути оголошені ще до їхнього використання. Основна форма оголошення змінної має вигляд:

*тип ідентифікатор [=значення][,ідентифікатор[=значення]…];*

Під типом розуміють один із простих типів *Java*, ім'я класу або інтерфейсу. Деякі приклади оголошення змінних у *Java*:

*int a,b,c; int d=2, e, f=5; byte z=22; double pi=3.1415926;*

*char x;*

***Динамічна ініціалізація.*** *Java* допускає динамічну ініціалізацію змінних не тільки константами, але й виразами. *// Демонстрація динамічної ініціалізації змінних class DynInit {*

*public static void main(String args[ ]){ double a=3.0,b=4.0; // звичайна ініціалізація*

*//оголошення і динамічна ініціалізація double c=Math.sgrt (a\*a+b\*b);*

*System.out.println ("Довжина гіпотенузи"+с);*

*}*

*}*

Вираз для ініціалізації змінної може використовувати будьякий елемент, доступний в момент ініціалізації: виклик методу, використання змінних або літералів. У нашому прикладі використано метод *sgrt* з класу *Math* у виразі для ініціалізації змінної *c*.

***Область доступності і час існування змінних.*** У *Java* можна оголошувати змінні не тільки на початку методу, (наприклад, *main()*), але й всередині будь-якого блоку. Блок − це частина програмного коду, яка починається з відкриваючої фігурної дужки { і закінчується закриваючою фігурною дужкою }. Усередині блоку змінна може бути оголошена в будь-якому місці, але до її першого використання.

Залежно від того, де оголошено змінну, задається область доступності (видимості) до неї. В *Java* область доступності змінної визначається класом, методом і блоком. Область доступності, яку визначають класом, розглядатимемо далі. Тепер розглянемо області, які визначають всередині методу. Область доступності, яку визначають методом, розпочинається з фігурної дужки. Якщо метод має параметри, то вони також доступні в усьому методі. Якщо змінну оголошено всередині блоку, вона є недоступною ззовні цього блоку. Отже, ця змінна є захищена від несанкціонованого доступу і зміни. Правила визначення області доступності забезпечують фундамент для інкапсуляції.

Області доступності можуть бути вкладеними. Це означає, що об'єкти, оголошені в зовнішній області, будуть доступні для коду внутрішнього блоку. Але об'єкти, оголошені у внутрішньому блоці, не будуть доступними ззовні.

// *Демонстрація області доступності змінної блоку class Scope { public static void main(String args[ ]){ int x; // змінна доступна всюди в main x=10;*

*if(x==10) { //початок нового блоку*

*int y=20; //змінна у доступна тільки в цьому блоці*

*System.out.println ("x і y" + x + " " + y);*

*x=y+2;*

*} // завершення блоку*

*// y=100; Помилка! Змінна y вже недоступна*

*System.out.println("x є" + x);*

*} // завершення методу*

*}// завершення класу*

Важливо також пам’ятати, що змінні створюються в момент входу в їхню область доступності і знищуються під час виходу з неї. Отже, змінні не зберігають своїх значень при повторному звертанні до методу і т. п.

У мові *Java* не можна оголошувати з одним іменем змінну в укладених блоках (у мові *С++* це можливо). Наприклад, така програма не буде правильно компілюватися:

*class ScopeErr {*

*public static void main(String args[ ]){ int bar=1;*

*{*

*int bar=2; //Помилка! bar вже оголошена вище*

*}*

*}*

*}*

#### 2.1.4. Перетворення і приведення типів

У процесі програмування присвоєння значень одного типу змінній іншого типу є звиклим явищем. Якщо два типи сумісні, в *Java* автоматично виконається таке перетворення. Проте не всі типи сумісні і тому не всі перетворення типів дозволені в неявному виді.

***Автоматичне перетворення типів.*** У *Java* можливе автоматичне перетворення типів за умови виконання двох умов:

1. два типи сумісні;
2. тип призначення більший від вихідного типу.

При виконанні цих двох умов виконується перетворення з розширенням (*widening conversion*). Наприклад, змінної типу *int* достатньо, щоб прийняти всі значення типу *byte*.

При перетворенні з розширенням числові типи сумісні одні з одними, проте несумісні з *char* і *boolean*. Крім того, *char* і *boolean* несумісні між собою.

***Приведення несумісних типів.*** Перетворення типу *int* у *byte* не виконається автоматично. Цей вид перетворення іноді називають перетворенням зі звуженням (*narrowing conversion*), тому що значення скорочується, щоб підійти під тип призначення. Для перетворення між двома несумісними типами необхідно виконати операцію приведення. Приведення (*cast*) − це перетворення типів у явному вигляді. Воно матиме загальний вигляд:

*(тип-призначення) значення,* де *тип-призначення* задає тип, до якого необхідно перетворити значення.

Під час приведення типу *int* до типу *byte*, якщо значення цілого більше діапазону представлення *byte*, воно буде вкорочене за модулем діапазону типу *byte* (тобто буде взято залишок від ділення цілого на діапазон типу *byte*).

Внаслідок присвоєння дійсного числа цілому виконується інший тип перетворення: відсікання (*trancation*), тобто відкидається дробова частина. Якщо цілочисельна частина перевищуватиме можливе значення для цілого типу, то вона буде вкорочена за модулем діапазону типу призначення. *// Демонстрація приведення типів class Conversion {*

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

*int i=257; double d=323.142;*

*b=(byte)i;*

*System.out.println("і в b :" + і +" "+ b); i=(int)d;*

*System.out.println("*d *в* i *:" +* d *+" "+* i*); b=(byte)d;*

*System.out.println("*d *в* b *:" +* d *+" "+ b);*

*}*

*}*

Результат роботи цієї програми матиме вигляд: *i* в *b* : 257 1 *d* в *i*: 323.142 323 *d* в *b* : 323.142 67

Аналізуючи ці результати необхідно пам’ятати, що діапазон для типу *byte* є 256.

***Автоматичне перетворення типів у виразах.*** У мові *Java* визначено декілька правил перетворення типів у виразах:

− змінні типу *byte* і *short* завжди автоматично перетворюються в *int;*

− якщо один з операндів має тип *long*, увесь вираз перетворюється в *long;*

− якщо один операнд має тип *float*, то увесь вираз перетворюється у *float;*

− якщо один операнд має тип *double*, то результат буде типу *double*.

Незважаючи на вигідність автоматичного перетворення типів у виразах, воно може спричинити помилки компілятора. Наприклад, коректний на перший погляд код може бути джерелом проблеми:

*byte b=50;* b=b\*2; //*Помилка!* *Тип* int *не присвоюється* byte.

Виявляється, що значення 100 цілком допустиме для *byte* не може бути збережене в *b*. Внаслідок обчислень операнди були автоматично перетворені в тип *int*, результат також типу *int*. Без виконання операції приведення його не можна присвоїти типу *byte*. Це є справедливим навіть у тому випадку, коли значення належить діапазону призначення.

У випадках, коли виникають сумніви, краще задати явне приведення типів. Коректний код має вигляд:

*bуte b=50; b=(byte)b\*2;*

#### 2.1.5. Масиви

Масив (*array*) – це група однотипних елементів, які "відгукуються" на одне спільне ім’я. Масиви можна створювати будьякого типу і довільного розміру. Доступ до заданого елемента масиву здійснюється за допомогою індексів.

Організація масивів у мові *Java* інша, ніж у *С++*.

***Одновимірні масиви.*** Формат оголошення одновимірного масиву має вигляд:

*тип ім’я\_змінної[];*

Таке оголошення засвідчує, що *ім’я\_змінної* – це змінна масиву, проте вона насправді ще не існує і має значення *null*. Для того, щоб зв’язати *ім’я\_змінної* з реальним фізичним масивом, необхідно виділити пам’ять за допомогою оператора *new* *ім’я\_змінної=new тип[розмір];*

Елементи масиву в *Java* автоматично ініціалізуються нулями. Усі індекси масиву розпочинаються з нуля. Можна об’єднати оголошення змінної масиву з виділенням пам’яті:

*int month\_days[ ]=new int[12];*

Масиви можна ініціалізувати аналогічно як і прості змінні. У цьому випадку розмір масиву визначається автоматично кількістю елементів ініціалізації. Ініціалізатор масиву – це список виразів, розділених комою і розміщених у фігурних дужках.

*// Демонстрація ініціалізації масиву*

*class AutoArray{*

*public static void main(String args[ ]){ int month\_days[ ]={31,28,31,30,31,30,31,31,30,31,30,31}*

*}*

*}*

Компілятор *Java* стежить за індексами масиву, щоб не вийшли за межі масиву. Виконавча система *Java* обов’язково перевірить усі індекси масивів. У випадку виходу за межі видасть помилку виконання. Це також відрізняє мову *Java* від мови *С++*.

***Багатовимірні масиви.*** У мові *Java* багатовимірні масиви (як і в *С++*) є насправді масивами масивів. Щоб оголосити багатовимірний масив, необхідно задати додатковий індекс за допомогою ще одного набору квадратних дужок:

*int twoD[ ][ ]=new int[4][5];*

Другий індекс визначає стовпець, перший рядок. Виокремлюючи пам’ять для багатовимірних масивів, необхідно визначити пам’ять тільки для першого виміру. Виділення пам’яті для решти вимірів можна здійснити окремо. У цьому випадку кількість елементів у кожному рядку може бути різною. Наступний програмний код створює двовимірний масив для трикутної матриці. *// Демонстрація двовимірного масиву*

*class TwoDAgain { public static void main(String args[ ] ){*

*int twoD[ ][ ]=new int [4][ ]; twoD[0]=new int [1]; twoD[1]=new int [2]; twoD[2]=new int [3]; twoD[3]=new int [4];*

*}*

*}*

Багатовимірні масиви ініціалізуються даними між двома парами фігурних дужок. Напиклад:

*class Matrix{ public static void main(String args [ ]){ double m[ ][ ]= {*

*{0, 1, 2, 3}*

*{4, 5, 6, 7}*

*{8, 9, 10, 11}*

*{12, 13, 14, 15}*

*};*

*};*

*};*

***Альтернативний синтаксис оголошення масивів.*** Існує альтернативний формат оголошення масиву *тип[] ім’я\_змінної;*

Такі оголошення є еквівалентними:

*int a1[ ]=new int[3]; int[ ] a2=new int[3]; char twod1[ ][ ]=new char[3][4]; char [ ][ ] twod2=new char[3][4];*

**Зауваження.** Оголошення масивів оператором *new* вказує на те, що масиви в мові *Java* – це об’єкти. Число елементів, які можна розмістити в масиві, зберігається в екземплярі змінної *length*. Наприклад, *a1.length* дорівнює 3.