### 2.2. Операції

Операції в мові *Java* можна розділити на такі групи:

− арифметичні;

− порозрядні;

− логічні;

− операції відношення.

Далі у тексті наведемо зведені таблиці цих операцій. Детальнішу інформацію можна подивитися в [9].

#### 2.2.1. Арифметичні операції

Арифметичні операції використовуються при програмуванні математичних виразів. Їхній перелік наведено в табл. 2.4.

#### 2.2.2. Побітові операції

У мові *Java* визначено декілька побітових операцій (табл. 2.5), які застосовують до цілочисельних типів (*long, int, short, char* i *byte*). Такі операції виконуються над конкретними бітами своїх операндів.

Усі цілочисельні типи представлено степенями числа 2. Наприклад, *42* = *2* + *8* + *32* = *21* + *23* + *25* = *00101010* . Від’ємні числа в мові *Java* представляють як доповнення до 2. Тобто від’ємні числа спочатку піддають інверсії (1 заміняються на 0, 0 – на 1), а потім додають 1 (*мінус сорок два* записують як 11010101, *плюс один* дає 11010110). Найстарший розряд визначає знак цілого.

Т а б л и ц я 2.4. **Арифметичні операції**

|  |  |
| --- | --- |
| Операція | Результат |
| + | Додавання |
| - | Віднімання (а також унарний мінус - від’ємне значення) |
| \* | Множення |
| / | Ділення |
| % | Ділення за модулем (залишок від ділення, цілі і дійсні) |
| ++ | Інкремент (збільшення на одиницю операнда, є постфіксна і префіксна форми) |
| += | Присвоєння результату додавання (a=a+2 еквівалентно a+=2) |
| -= | Присвоєння результату віднімання |
| \*= | Присвоєння результату множення |
| /= | Присвоєння результату ділення |
| %= | Присвоєння результату ділення за модулем |
| -- | Декремент (зменшення на одиницю операнда, є постфіксна і префіксна форми) |

Т а б л и ц я 2.5. **Побітові операції**

|  |  |
| --- | --- |
| Операція | Результат |
| ~ | Порозрядне унарне заперечення (NOT) |
| & | Порозрядне "І" (AND) |
| | | Порозрядне АБО (OR) |
| ^ | Порозрядне заперечення АБО (OR) |
| >> | Зсув вправо |
| >>> | Зсув вправо з заповненням нулем |
| << | Зсув вліво |
| &= | Порозрядне присвоєння результату I |
| | = | Порозрядне присвоєння результату АБО |
| ^= | Порозрядне присвоєння результату заперечуючого АБО |
| >>= | Порозрядне присвоєння результату зсуву вправо |
| >>>= | Порозрядне присвоєння результату зсуву вправо з заповненням нулем |
| <<= | Порозрядне присвоєння результату зсуву вліво |

#### 2.2.3. Операції відношення

У мові *Java* за допомогою перевірки на рівність, нерівність тощо (табл. 2.6) можна порівнювати значення будь-якого типу: цілі, числа з плаваючою комою, символи, логічні типи.

Т а б л и ц я 2.6. **Операції відношення**

|  |  |
| --- | --- |
| Операція | Результат |
| == | Рівність |
| != | Нерівність |
| > | Більше ніж |
| < | Менше ніж |
| >= | Більше або рівне |
| <= | Менше або рівне |

**Зауваження 1.** Для об’єктів оператор "==" перевіряє, чи вказують покажчики на один об’єкт, а не рівність двох об’єктів. Для перевірки на рівність (співпадання) об’єктів є метод *equals().*

**Зауваження 2.** У мові *С++* будь-яка відмінна від нуля величина має значення *true, 0 - false*. Тому в *С++* правильним буде код:

*int done;*

*...*

*if (!done) { };*

Мовою *Java* необхідно писати у таких випадках тільки з явною перевіркою на рівність нулю:

*int done;*

*...*

*if (done!=0) {};*

#### 2.2.4. Логічні операції

Логічні операції виконують тільки з операндами типу *boolean*. Їхній перелік наведено у табл. 2.7. Логічні операції виконують над значеннями типу *boolean* аналогічно, як вони діють над бітами цілих типів.

Перевагу використання логічних обчислень за скороченою схемою демонструє фрагмент програми:

*if(denom!=0 && num/denom>10)*.

Т а б л и ц я 2.7. **Логічні операції**

|  |  |
| --- | --- |
| Операція | Результат |
| & | Логічне I |
| | | Логічне АБО |
| ^ | Логічне заперечуюче АБО |
| || | Обчислення АБО за короткою схемою |
| && | Обчислення І за короткою схемою |
| ! | Логічне упарне НІ |
| &= | Присвоєння результату логічного І |
| |= | Присвоєння результату логічного АБО |
| ^= | Присвоєння результату логічного заперечуючого АБО |
| == | Рівність |
| != | Нерівність |
| ?: | Тернарний оператор if-then-else |

#### 2.2.5. Операція присвоєння

Операцію присвоєння виконують у мові *Java* аналогічно як і в усіх інших мовах програмування. Вона має такий вигляд:

*змінна = вираз;*

Тип змінної повинен бути сумісним з типом виразу. Ця операція дає змогу створювати ланцюжок присвоєнь: *x=y=z=100;*

#### 2.2.6. Умовна операція «?:»

Загальна форма (тернарної, тобто операції, яка має три операнди):

*вираз1? вираз 2: вираз 3*

У цій формі *вираз 1* повинен повертати тип *boolean*. Якщо *вираз 1* істинний, то обчислюють *вираз 2*, інакше *вираз 3*. *Вираз 2* і *вираз 3* повинні повертати значення одного типу. Використання цієї операції демонструє наступний код:

*// Отримання абсолютного значення числа і*

*clas Ternary {*

*public static void main(String args[ ]){ int i,k; i=-10; k=i<0?-i:i;*

*}*

*}*

#### 2.2.7. Пріоритети операцій

У мові *Java* погоджено такий порядок виконання операцій у виразах:

1. ( ) [] .
2. ++ -- ~ !
3. \* / %
4. + -
5. >> >>> <<
6. > >= < <=
7. == !=
8. &
9. ^
10. |
11. && 12. ||
12. ?:
13. = операція=

Бажано використовувати круглі дужки з метою спрощення програми, тому що їх наявність не впливає на швидкість виконання програми. Наприклад, не викликає сумніву, який з цих двох виразів зручніше прочитати:

*a|4+c>>b&7||b>a%3* чи *(a|(((4+c)>>b)&7))||(b>(a%3))*

### 2.3. Оператори

Оператор − це певною мірою довільна стрічка програмного коду, яка закінчується крапкою з комою. Оператором може бути вираз, виклик методу, оголошення. Об’єднані фігурними дужками в блок оператори утворюють складений оператор.

Послідовне виконання коду можна змінювати, використовуючи оператори керування. У мові *Java* маємо три категорії операторів керування: вибору, ітерації та переходу.

#### 2.3.1. Оператори вибору

Виконання тієї чи іншої частини коду програми залежно від виконання певної умови організовується за допомогою оператора *if*. Його загальний вигляд:

*if (умова) оператор 1; else оператор 2;*

*Оператор 1, оператор 2* – це не тільки один оператор, але й складений оператор. Варіанта *else* може не існувати. Оператори *if* можуть бути вкладеними. Вони можуть утворювати ланцюжок операторів:

*if (умова) оператор; else if (умова) оператор; else if (умова) оператор;*

*… else оператор;*

Якщо одна з умов стає *true*, то виконується оператор, який є наступним за цим *if*. Решта операторів пропускають.

Подібним до оператора *if* є оператор *switch*, який дає змогу здійснювати розгалуження програмного коду в декількох напрямах. Він має вигляд:

*switch (вираз)*

*{*

*case значення 1:*

*//послідовність операторів break; case значення 2:*

*//послідовність операторів break;*

*... case значення N:*

*//послідовність операторів break; default:*

*//послідовність операторів break;*

*}*

*Вираз* може повертати значення будь-якого простого типу. Кожне значення, задане в операторах *case*, повинне мати тип, сумісний з типом *виразу*. Значення в *case* повинні бути літералами, а не змінними. Оператори *case* з однаковими значеннями недопустимі.

Оператор *switch* працює так: після обчислення значення виразу в заголовку оператора порівнюють його результат послідовно зі значеннями усіх констант в усіх варіантах *case*. Виконується послідовність операторів, які йдуть за варіантом *case*, що відповідає значенню виразу. Якщо ні одна з констант не збігається зі значенням виразу, то виконується послідовність операторів, заданих за замовчуванням (після *default*). Оператор *default* може бути відсутнім.

Оператор *break* є ознакою закінчення послідовності операторів і використовується для виходу з оператора, який безпосередньо його містить. У нашому випадку він передає керування на перший рядок за фігурною закриваючою дужкою оператора *switch*. Оператор *break* є необов’язковим. Якщо його пропустити, то виконаються усі наступні оператори *case*, доки не зустрінеться оператор *break* або не буде завершення оператора *switch*.

Оператори *switch* можна використовувати в складі операторів, які належать до іншого оператора *switch*. Так як *switch* визначає свій внутрішній блок {}, між операторами *case* внутрішнього і зовнішнього *switch* не виникає конфліктів.

Три важливі особливості оператора *switch*:

1. на відміну від оператора *if* оператор *switch* може здійснювати перевірку тільки на рівність;
2. не допускається в одному операторі *switch* двох однакових значень в *case;*
3. оператор *switch* є ефективнішим, ніж набір (ланцюг) операторів *if*.

#### 2.3.2. Ітераційні оператори (циклу)

Ітераційні оператори (у мові *Java* − *for, while, do-while*) дають змогу створювати структури, які називаються циклами (*loop*). Цикли, або ітераційні оператори, дають змогу повторювати виконання операторів або груп операторів. Число повторень у деяких випадках фіксоване, а в інших визначається під час обчислень внаслідок перевірки умов виходу з циклу.

Оператор *while* є базовим оператором циклу в мові *Java*. Він повторює виконання оператора або складеного оператора до того часу, доки керуючий вираз (умова) є істинним. Синтаксис циклу *while* такий :

*while (умовний вираз) {*

*// тіло циклу*

*}*

Якщо в циклі виконується один оператор, фігурні дужки не обов’язкові. Якщо значення умовного виразу є *false*, то тіло циклу не виконуватиметься жодного разу. Тіло циклу *while* може бути пустим. Такий цикл використовують в наступному коді програми.

*//Демонстрація циклу clаss NoBody { public static void main{String args[ ]){*

*int i,j;*

*i=100; j=200;*

*// знаходження середнього між i і j while (++і < --j);*

*System.out.println ("середнє рівне*

*" + i);*

*}*

*}*

Зрозуміло, що в циклі *while* перевірка умови здійснюється перед виконанням тіла циклу. Можлива ситуація, коли тіло циклу не виконується жодного разу. Інколи необхідно виконати тіло навіть тоді, коли умова не виконується. У цих випадках використовують оператор циклу *do-while.* Синтаксис оператора:

*do {*

*// тіло циклу*

*} while (умовний вираз);*

У циклі *do-while* перевірка умови здійснюється після виконання тіла циклу. Тому у випадку використання оператора *dowhile* тіло циклу виконується хоча б один раз.

Ще однією формою оператора циклу в *Java* є *for.* Конструкція цього оператора виглядає так:

*for (ініціалізація; умова; ітерація)*

*{*

*// тіло циклу*

*}*

Цикл *for* працює так. Перед початком роботи виконуються дії, вказані в частині ініціалізації. Найчастіше − це ініціалізація змінної керування циклом, яка виконує роль лічильника, що керує роботою циклу. Ініціалізаційний вираз виконується тільки один раз. Далі перевіряється умова, яка є булевим виразом. Переважно тут порівнюється змінна керування циклом з її кінцевим значенням. Якщо умова є істинна, цикл виконується, якщо – *false*, то тіло циклу не виконується. Після цього виконується частина циклу, яку називають ітерацією. Переважно – це вираз, де змінна циклу збільшується або зменшується на одиницю. Потім виконується властиво цикл, протягом якого внаслідок кожного проходження спочатку обчислюється умовний вираз, потім виконується тіло циклу, а після цього – ітераційний вираз. Цей процес повторюється доти, доки умовний вираз не стане *false*. Цикли *for* можуть бути вкладеними.

Часто змінну циклу використовують тільки в цьому циклі і більше ніде. У цьому випадку можна оголосити цю змінну в ініціалізаційній частині оператора *for*. Наприклад,

*// Оголошення змінної керування циклом всередині for class ForTick { public static void main(String args[ ]){ for (int n=10; n>0; n--)*

*System.out.println("Обернений рахунок-"+n);*

*}*

*}*

Інколи необхідно, щоб в ініціалізаційній та ітераційній частинах циклу *for* виконувалось не по одному оператору. В цьому випадку ці оператори треба розділити комою. Наприклад, *// Використання ком в операторі циклу class Comma {*

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

*int a,b;*

*for (a=1, b=4; a<b; a++, b--){*

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

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

*}*

*} }*

Існує декілька модифікацій циклу *for* тому, що кожна з трьох або всі три частини циклу – ініціалізаційна, перевірочна та ітераційна – не обов’язково використовуються для вказаних цілей і взагалі можуть бути відсутніми, проте їхнє розділення крапкою з комою (;) опускати не можна. Наприклад, організація безмежного циклу виглядатиме так: *for ( ; ; ){…};*

#### 2.3.3. Оператори переходу

У мові *Java* передбачено три оператори переходу: *break, continue, return*. Вони призначені для передавання керування іншій частині програми. Є ще один засіб змінити послідовність виконання операторів у програмі – це обробка виняткових ситуацій. Оператор *break* використовують у мові *Java* у трьох випадках:

− вихід з оператора *switch;*

− вихід з циклу;

− він є "цивілізованим" оператором *goto*.

За допомогою оператора *break* можна вийти з циклу примусово, проігнорувавши умовний вираз і всі оператори, які йдуть за *break*. Керування передається наступному за тілом циклу оператору. Оператор *break* можна використати у будь-якому циклі, навіть у нескінченному. Якщо *break* використовується у вкладених циклах, то вихід відбувається тільки з внутрішнього (одного) циклу. У циклі може бути декілька операторів *break*.

Оператори *break*, які належать оператору *switch*, що є в тілі циклу, не впливають на хід виконання циклу.

**Зауваження.** Оператор *break* не є нормальним виходом з циклу. Для цього існує умовний вираз. Його використання в циклах бажане тільки в особливих випадках.

Часто буває необхідно вийти з глибоко вкладеного циклу. Для реалізації такої ситуації в *Java* визначено розширену форму оператора *break* :

*break мітка;*

Тут *мітка* – це ім’я, яке ідентифікує блок програми. Під час виконання цієї форми *break* керування передається блоку з міткою, який також повинен мати у своєму складі оператор *break*. Блок, на який передається керування не обов’язково повинен містити блок, з якого здійснюється вихід.

Для присвоєння блокові імені необхідно на його початку поставити мітку (label) – це будь-який ідентифікатор, дозволений в *Java*, після якого стоїть двокрапка (":"). При посиланні на мітку блоку з оператора *break*, керування буде передано в кінець блоку. Найчастіше *break* з міткою використовується для виходу з укладених циклів.

Оператор *continue* можна використовувати у будь-якій з трьох форм циклів. Його виконання спричиняє таку зміну логіки програми, що решта операторів тіла циклу пропускаються. Для циклів *while* або *for* відразу ж за оператором *continue* розпочинається новий крок, а для циклу *do-while* перевіряється умова на виході. Проте для усіх циклів частина коду від *continue* до кінцевого оператора циклу пропускається. Як і *break* оператор *continue* може мати мітку, яка визначає, на який із вкладених циклів здійснюється перехід. Наприклад,

*// Використання оператора continue з міткою class ContinueLabel {*

*public static void main(String args[ ]){ outer : for (int i=0; i<10; i++){*

*for (int j=0; j<10; j++){ if (j>i) {*

*System.out.println(); continue outer; }*

*System.out.print(" "+(i\*j)); } }*

*System.out.println();*

*}*

*}*

Ця програма виводить трикутну таблицю множення чисел від

0 до 9.

Для явного повернення з методу використовують оператор *return*.