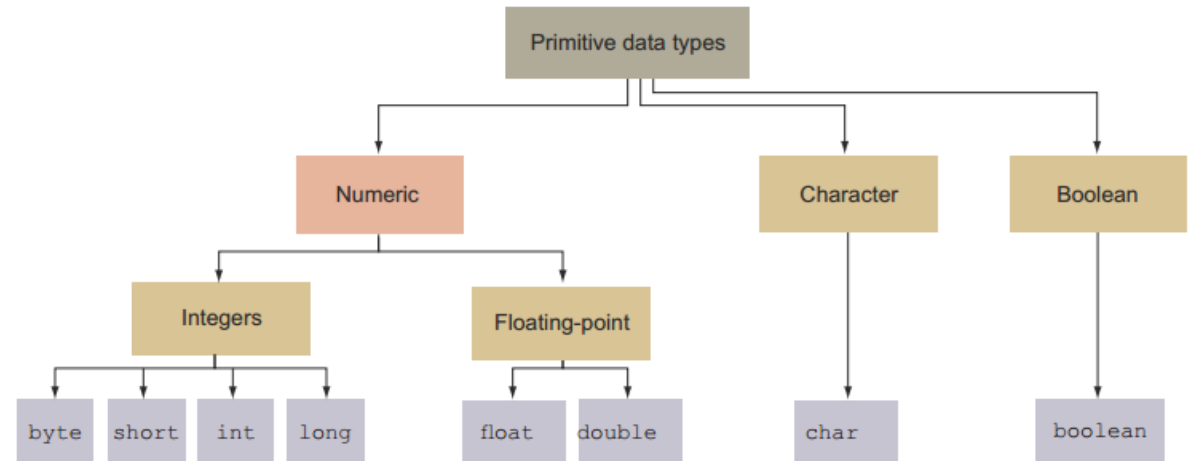
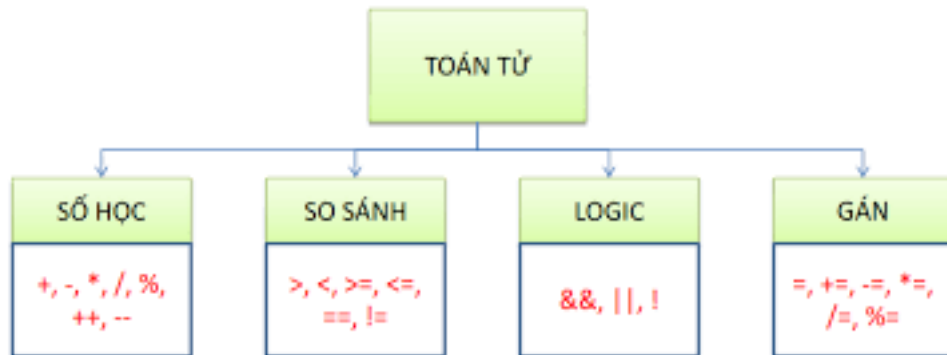


Chương 02: GIỚI THIỆU VỀ JAVA

PHÉP TOÁN VÀ KIỂU DỮ LIỆU



GVHD: ThS. Hồ Khôi

Email: khoihue@gmail.com

Cấu trúc chương trình đơn giản

- Bao gồm 2 phần

- Khai báo thư viện, module:

import <tên thư viện>;

package <tên module, gói thư viện>;

- Class & hàm main

public class <tenClass>{

public static void main(String[] args)

 {

 *//--- Mã nguồn dưới dạng các lệnh cần thực thi*

 }

}

Cấu trúc chương trình

Khai báo class với từ khóa public

Khai báo: package, import, ...

```
1 import java.util.*;
2
3 public class HelloWorld {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7         System.out.println("Hello World");
8         System.out.println("Hello World");
9     }
10
11 }
12
```

public class phải có hàm main, được khai báo public static void

Lệnh được thực hiện dựa vào Class, Object & Method. Trong trường hợp trên, ta gọi đối tượng out của lớp System, và thông qua phương thức println để xuất chuỗi ra màn hình

System.out.printf

System.out.printf là lệnh dùng cho việc in dữ liệu ra màn hình máy tính.

Cú pháp:

System.out.printf("Chuỗi thông báo", tham số, ...);

- "*Chuỗi thông báo*": Thông tin cần xuất ra trên màn hình
- Tham số, ... : Các giá trị dữ liệu cần đưa vào, thể hiện trên chuỗi thông báo

System.out.print

Đây là phương thức xuất chuẩn của java, dùng để cho phép in ra thông báo trên màn hình. Tuy nhiên, tham số của hàm này khác với printf ở chỗ: các đối tượng được ghé nối với nhau bằng toán tử “+”

Cú pháp:

System.out.print(tham số + tham số + ...);

| Định dạng | Tham số | Kết quả | Ghi chú |
|-----------|---------|----------------|-------------------------------------|
| %d | i | “123” | độ rộng 3 |
| %05d | i | “00123” | Thêm 2 số 0 |
| %7o” | i | “ 123” | Hệ 8, canh phải |
| %-9x | i | “7b “ | Hệ 16, canh trái |
| %c | i | “{“ | Ký tự có mã ASCII 123 |
| %-#9x | i | “0x7b “ | Hệ 16, canh trái |
| %10.5f | x | “ 0.12346” | Độ rộng 10, có 5 chữ số thập phân |
| %-12.5e | x | “1.23457e-01 “ | Canh trái, in ra dưới dạng khoa học |

Ký tự đặc biệt – Escape characters

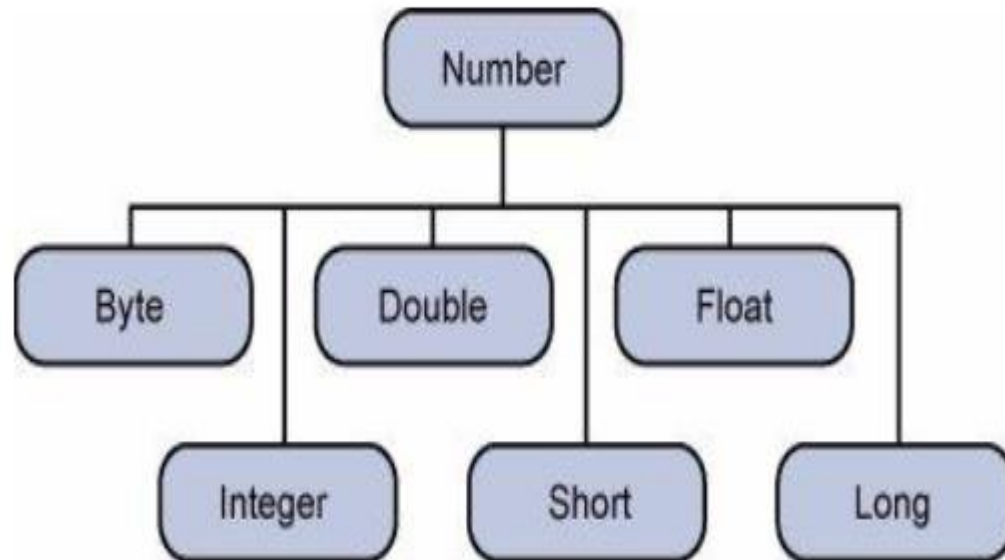
| Ký tự | Tác dụng | Mã ASCII |
|-------|--|----------|
| \n | Xuống hàng mới | 10 |
| \t | Tab | 9 |
| \b | Xóa ký tự bên trái | 8 |
| \r | Con trỏ trở về đầu hàng | 13 |
| \f | Sang trang | 12 |
| \a | Phát tiếng còi | 7 |
| \\ | Xuất dấu chéo ngược | 92 |
| \' | Xuất dấu nháy đơn ‘ | 39 |
| \'' | Xuất dấu nháy kép “ | 34 |
| \xdd | Xuất ký tự có mã ASCII dạng Hex là dd | |
| \ddd | Xuất ký tự có mã ASCII dạng Dec là ddd | |
| \0 | Ký tự NULL | 0 |

Kiểu dữ liệu

- Java cung cấp 8 kiểu dữ liệu nguyên thủy “***primitive data***”, hay còn gọi là built-in data type. Bao gồm:
 - 4 kiểu dùng cho chứa số nguyên: **byte, short, int, long**
 - 2 kiểu chứa số thực, có dấu chấm động: **float, double**
 - 1 kiểu chứa dữ liệu logic : **boolean**
 - 1 kiểu chứa ký tự: **char**

Ví dụ Kiểu số liệu Number

- Integer, Long, Byte, Double, Float, Short



Ví dụ Code Java khai báo kiểu dữ liệu

```
public class Test {  
  
    public static void main(String args[]) {  
        Integer x = 5; // boxes int to an Integer object  
        x = x + 10;    // unboxes the Integer to a int  
        System.out.println(x);  
    }  
}
```

Output

15

Primitive data types

| Type | Description | Default | Size | Example Literals |
|---------|-------------------------|---------|---------|---|
| boolean | true or false | false | 1 bit | true, false |
| byte | twos complement integer | 0 | 8 bits | (none) |
| char | Unicode character | \u0000 | 16 bits | 'a', '\u0041', '\101', '\\', '\'', '\n', '\B' |
| short | twos complement integer | 0 | 16 bits | (none) |
| int | twos complement integer | 0 | 32 bits | -2, -1, 0, 1, 2 |
| long | twos complement integer | 0 | 64 bits | -2L, -1L, 0L, 1L, 2L |
| float | IEEE 754 floating point | 0.0 | 32 bits | 1.23e100f, -1.23e-100f, .3f, 3.14F |
| double | IEEE 754 floating point | 0.0 | 64 bits | 1.23456e300d, -1.23456e-300d, 1e1d |

Range of numeric data types in Java

| Type | Size | Range |
|--------|---------|--|
| byte | 8 bits | -128 .. 127 |
| short | 16 bits | -32,768 .. 32,767 |
| int | 32 bits | -2,147,483,648 .. 2,147,483,647 |
| long | 64 bits | -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 |
| float | 32 bits | $3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$ |
| double | 64 bits | $1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$ |

Biến & hằng – (Variable - Constant)

✓ Biến – Variable

Biến được xem là những đối tượng khai báo trong chương trình để lưu trữ dữ liệu thuộc một “*kiểu xác định*” nào đó (Theo quy ước của người lập trình). Nội dung của biến có thể bị thay đổi tùy thuộc vào thuật toán & thời điểm mà ta truy xuất đến. Cú pháp khai báo biến trong C được mô tả như sau:

<kiểu_dữ_liệu> <ten_bien>;

<kiểu_dữ_liệu> <ten_bien> = <giá_trị>;

Ví dụ:

int n, s; [1]

short bcc = 5; [2]

long gt = 0, i, a = 12, b = 7, c = -11; [3]

Ghi chú:

- ❖ [1] – Khai báo 2 biến n và s thuộc kiểu nguyên, độ lớn mỗi biến là 2 bytes, có thể chứa giá trị trong khoảng từ -32768 ... + 32767
- ❖ [2] – Khai báo biến bcc thuộc kiểu short và khởi gán giá trị ban đầu là 5
- ❖ [3] – Khai báo các biến gt, i, a, b, c kiểu long và tiến hành khởi gán giá trị ban đầu cho các biến gt, a, b, c

Biến & hằng – (Variable - Constant)

✓ Hằng – Constant

Hằng là đối tượng được khai báo trong chương trình để lưu trữ một giá trị thuộc một “*kiểu xác định*” nào đó (*kiểu dữ liệu của hằng sẽ được ngôn ngữ xác định tùy thuộc vào giá trị khai báo cho nó*). Nội dung của hằng **không thể** bị thay đổi trong suốt quá trình chương trình hoạt động.

Cú pháp khai báo hằng trong C có thể thực hiện bằng 1 trong 2 cách

final <kiểu_dữ_liệu> <ten_hằng> = <giá_trị>;

Ví dụ:

const int tyGia = 21500; [1]

const char kt = 'A'; [2]

Ghi chú:

- ❖ [1] – Khai báo hằng tyGia chứa giá trị 21500 trong chương trình (*Kiểu nguyên*)
- ❖ [2] – Khai báo hằng kt chứa chữ A (*Kiểu char*)

* Định danh – (*Identifier*)

- Đây là một khái niệm hết sức quan trọng trong chương trình, chương trình của mình, lập trình viên cần phải đặt tên cho thành phần (kiểu dữ liệu, ...), sau khi đã đặt tên, bạn có thể gọi sử dụng chúng. Tuy nhiên, việc đặt tên cho các thành phần trong chương trình phải tuân theo các quy tắc sau:
 - ❖ **Không được trùng tên, phải là duy nhất** (Nếu trùng thì sẽ gây nhầm lẫn).
 - ❖ Không được phép sử dụng các ký tự đặc biệt và khoảng trắng. Chỉ được dùng **chữ cái** [a,b,...z;A,B,...Z], **ký số** [0,1,...9] và **dấu gạch dưới** [_].
 - ❖ **Không được trùng tên với các từ khoá** quy ước trong ngôn ngữ lập trình.
 - ❖ **Luôn phải bắt đầu bởi ký tự trong bảng chữ cái hoặc dấu gạch dưới**.

| Variable Name | Valid/Invalid |
|-----------------|---|
| Employee | Valid |
| student | Valid |
| _Name | Valid |
| Emp_Name | Valid |
| @goto | Valid |
| static | Invalid as it is a keyword |
| 4myclass | Invalid as a variable cannot start with a digit |
| Student&Faculty | Invalid as a variable cannot have the special character & |

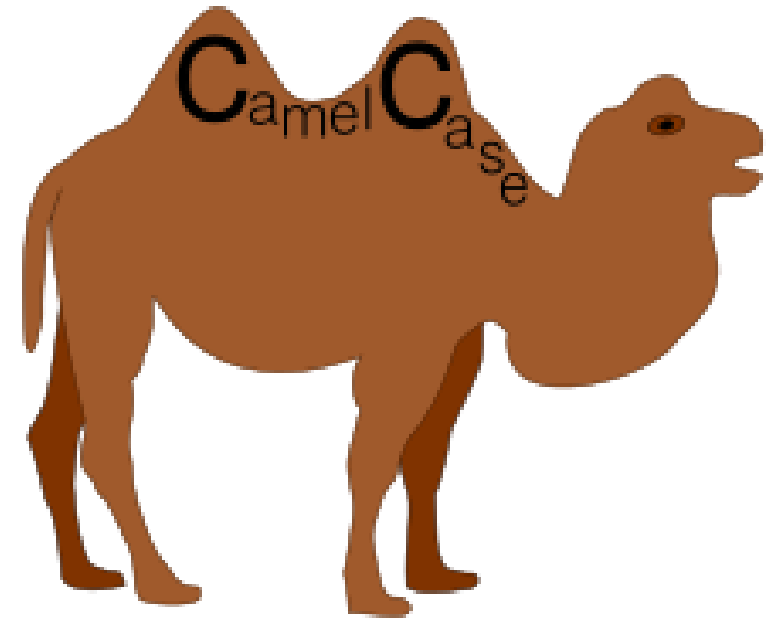
Quy ước đặt tên

Camel case

- Tuân theo nguyên tắc về đặt tên cho Identified trong lập trình java
- Bắt đầu các từ phải là viết chữ hoa, các ký tự còn lại phải là chữ thường, duy nhất từ đầu tiên phải viết chữ thường toàn bộ.

VD:

iPhone, eBay, fedEx, chieuDai, tyGia, ...



Đối tượng “scanner”

- **Tạo đối tượng: scanner**
- Phương thức nhận dữ liệu
 - **nextBoolean()**
 - **nextByte()**
 - **nextDouble()**
 - **nextFloat()**
 - **nextInt()**
 - **nextLine()**
 - **nextLong()**
 - **nextShort()**
- Đóng, giải phóng vùng nhớ
 - **close()**

Các phép toán

Để phục vụ cho việc tính toán (*Xử lý trong chương trình*), ngôn ngữ lập trình cung cấp các phép toán: Số học (*Arithmetic*), luận lý (*Logic*), Quan hệ (*Relative*), Gán (*Assignment*), ... các phép toán trong C thường được chia làm 2 dạng

- Phép toán một ngôi
- Phép toán hai ngôi

Trong đó, các thành phần tên gọi (*biến, hằng, hàm, ...*) tham gia trong phép toán được gọi là hạng thức (*hoặc toán hạng*), các kí hiệu dùng để biểu diễn phép toán được gọi là toán tử

Ví dụ:

- $a+b$: là một phép toán số học, trong đó **a**, **b** là các toán hạng và ký hiệu “+” là toán tử số học biểu diễn cho ý nghĩa của phép toán cộng

Lưu ý:

- Số ngôi của phép toán chính là số toán hạng tham gia trong phép toán tương ứng. Trong ví dụ trên, ta có phép toán hai ngôi

Phép toán số học – *Arithmetic operators*

| Operators | Description | Examples |
|-----------------------|---|------------|
| + (Addition) | Performs addition. If the two operands are strings, then it functions as a string concatenation operator and adds one string to the end of the other. | 40 + 20 |
| - (Subtraction) | Performs subtraction. If a greater value is subtracted from a lower value, the resultant output is a negative value. | 100 - 47 |
| * (Multiplication) | Performs multiplication. | 67 * 46 |
| / (Division) | Performs division. The operator divides the first operand by the second operand and gives the quotient as the output. | 12000 / 10 |
| % (Modulo) | Performs modulo operation. The operator divides the two operands and gives the remainder of the division operation as the output. | 100 % 33 |

Ví dụ về toán tử toán học

```
class ViDuToanTuToanHoc {  
    public static void main(String[] args) {  
  
        double so1 = 12.5, so2 = 3.5, ketQua;  
  
        // Cộng hai số  
        ketQua = so1 + so2;  
        System.out.println("so1 + so2 = " + ketQua);  
  
        // Trừ hai số  
        ketQua = so1 - so2;  
        System.out.println("so1 - so2 = " + ketQua);  
  
        // Nhân hai số  
        ketQua = so1 * so2;  
        System.out.println("so1 * so2 = " + ketQua);  
  
        // Chia hai số  
        ketQua = so1 / so2;  
        System.out.println("so1 / so2 = " + ketQua);  
  
        // lấy phần dư của so1 chia so2  
        ketQua = so1 % so2;  
        System.out.println("so1 % so2 = " + ketQua);  
    }  
}
```

Khi bạn chạy chương trình. Kết quả nhận được là:

```
so1 + so2 = 16.0  
so1 - so2 = 9.0  
so1 * so2 = 43.75  
so1 / so2 = 3.5714285714285716  
so1 % so2 = 2.0
```

Phép toán quan hệ - *relational operators*

| Relational Operators | Descriptions | Examples |
|----------------------|--|-----------|
| == | Checks whether the two operands are identical. | 85 == 95 |
| != | Checks for inequality between two operands. | 35 != 40 |
| > | Checks whether the first value is greater than the second value. | 50 > 30 |
| < | Checks whether the first value is lesser than the second value. | 20 < 30 |
| >= | Checks whether the first value is greater than or equal to the second value. | 100 >= 30 |
| <= | Checks whether the first value is lesser than or equal to the second value. | 75 <= 80 |

== toán tử so sánh sự bằng nhau

< toán tử so sánh nhỏ hơn

!= toán tử so sánh sự không bằng nhau

>= toán tử so sánh lớn hơn hoặc bằng

> toán tử so sánh lớn hơn

<= toán tử so sánh nhỏ hơn hoặc bằng

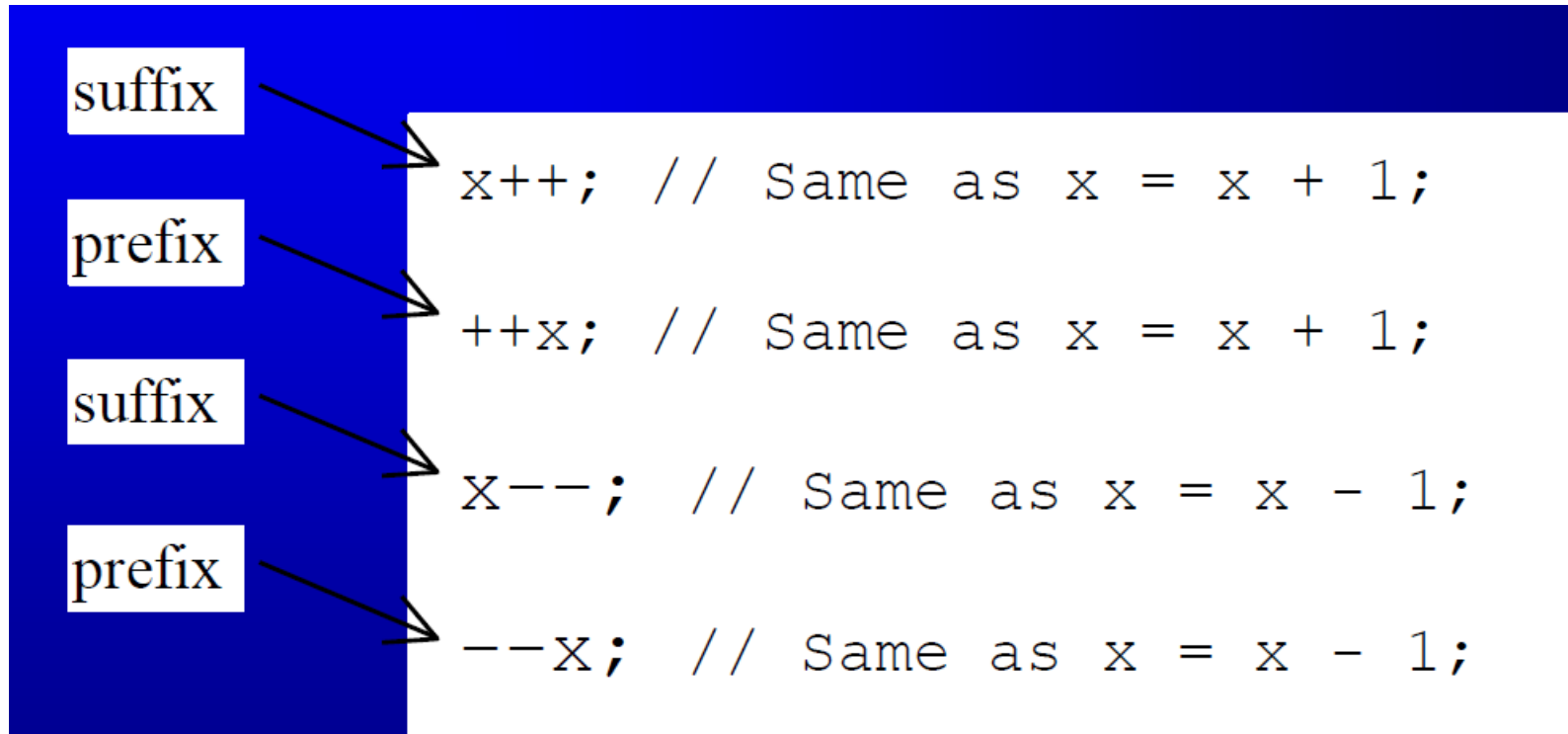
Phép toán quan hệ

```
class ViDuToanTuQuanHe {  
    public static void main(String[] args) {  
  
        int so1 = 5, so2 = 6;  
  
        if (so1 > so2){  
            System.out.println("Số 1 lớn hơn Số 2.");  
        } else{  
            System.out.println("Số 2 lớn hơn hoặc bằng Số 1");  
        }  
    }  
}
```

Khi bạn chạy chương trình. Kết quả nhận được là:

Số 2 lớn hơn hoặc bằng Số 1

Toán tử tăng, giảm — *Increment and Decrement operators*





Ví dụ về toán tử đơn phương:

```
class ViDuToanTuDonPhuong {  
    public static void main(String[] args) {  
  
        double so = 5.2, ketQua;  
        boolean giaTriSai = false;  
  
        System.out.println("Kết quả: +so = " + ++so);  
        // Kết quả: +so = 5.2  
  
        System.out.println("Kết quả: -so = " + --so);  
        // Kết quả: -so = -5.2  
  
        // ++so tương đương với so = so + 1  
        System.out.println("Kết quả: so++ = " + ++so);  
        // Kết quả: so++ = 6.2  
  
        // --so tương đương với so = so - 1  
        System.out.println("Kết quả: --so = " + --so);  
        // Kết quả: --so = 5.2  
  
        System.out.println("Ket qua: !giaTriSai = " + !giaTriSai);  
        // Ket qua: !giaTriSai = true  
    }  
}
```

```
int a = 5;  
++a    // a = 6  
a++    // a = 7  
--a    // a = 6  
a--    // a = 5
```

Các toán tử tăng và giảm (tiếp)

-  Sử dụng các toán tử tăng và giảm giúp các biểu thức ngắn gọn hơn, nhưng cũng làm cho chúng phức tạp và khó đọc hơn.
-  Nên tránh sử dụng các toán tử này trong những biểu thức làm thay đổi nhiều biến hoặc sử dụng cùng một biến nhiều lần như sau: `int k = ++i + i.`

Toán tử gán – *Assignment operators*

Toán tử gán (=) dùng để gán một giá trị vào một biến và có thể gán nhiều giá trị cho nhiều biến cùng một lúc.

| Toán tử | Miêu tả | Ví dụ |
|---------|---|--|
| = | Gán giá trị hạng phải cho hạng trái | A = B |
| += | Cộng 2 toán hạng rồi gán giá trị cho toán hạng trái | A += B tương đương A = A + B |
| -= | Trừ 2 toán hạng rồi gán giá trị cho toán hạng trái | A -= B tương đương A = A - B |
| *= | Nhân 2 toán hạng rồi gán giá trị cho toán hạng trái | A *= B tương đương A = A * B |
| /= | Chia lấy nguyên 2 toán hạng rồi gán giá trị cho toán hạng trái | A /= B tương đương A = A / B |
| %= | Chia lấy dư 2 toán hạng rồi gán giá trị cho toán hạng trái | A %= B tương đương A = A % B |
| <<= | Dịch trái toán hạng trái sang số vị trí bằng toán hạng phải rồi | A <<= B tương đương A = A << B |
| >>= | Dịch phải toán hạng trái sang số vị trí bằng toán hạng phải rồi | A >>= B tương đương A = A >> B |
| &= | Phép AND Bit | A &= B tương đương A = A & B |
| ^= | Phép OR loại trừ bit | A ^= B tương đương A = A ^ B |
| = | Phép OR bit | A = B tương đương A = A B |

Ví dụ toán tử gán

```
class ViDuToanTuGan {  
    public static void main(String[] args) {  
  
        int so1, so2;  
  
        // Gán 6 cho so1  
        so1 = 6;  
        System.out.println(so1);  
  
        // Gán giá trị của so1 cho so2  
        so2 = so1;  
        System.out.println(so2);  
    }  
}
```

Khi bạn chạy chương trình. Kết quả nhận được là:

6

6

Toán tử logic – *Logical operators*

Các toán tử logic làm việc với các toán hạng Boolean. Các toán tử quan hệ được sử dụng trong các cấu trúc điều khiển

| Toán tử | Mô tả |
|---------|---|
| && | Toán tử và (AND) Trả về một giá trị “Đúng” (True) nếu chỉ khi cả hai toán tử có giá trị “True” |
| | Toán tử hoặc (OR) Trả về giá trị “True” nếu ít nhất một giá trị là True |
| ^ | Toán tử XOR Trả về giá trị True nếu và chỉ nếu chỉ một trong các giá trị là True, các trường hợp còn lại cho giá trị False (sai) |
| ! | Toán tử phủ định (NOT) Toán hạng đơn tử NOT. Chuyển giá trị từ True sang False và ngược lại. |

Ví dụ

Toán tử logic thường xuyên được sử dụng trong vòng lặp.

```
class ViDuToanTuLogic {  
    public static void main(String[] args) {  
  
        int so1 = 1, so2 = 2, so3 = 9;  
        boolean ketQua;  
  
        // Ít nhất một biểu thức đúng để trả về giá trị true  
        ketQua = (so1 > so2) || (so3 > so1);  
        // Kết quả sẽ là true. Bởi vì so3 > so1  
        System.out.println(ketQua);  
  
        // Tất cả biểu thức phải đúng để trả về giá trị true  
        ketQua = (so1 > so2) && (so3 > so1);  
        // Kết quả nhận được là false vì biểu thức so1 > so2 bị sai  
        System.out.println(ketQua);  
    }  
}
```

```
true  
false
```

Chuyên kiểu, “Ép kiểu” – *Casting*

Khi khai báo biến để sử dụng trong chương trình của mình, đồng nghĩa với việc bạn “*Đề nghị được cấp phát một vùng nhớ dùng cho mục đích lưu trữ dữ liệu với không gian ... - tùy thuộc vào kiểu đã được khai báo*”, và như vậy chương trình của bạn đôi khi lại phải đối mặt với một số vấn đề khi xử lý. Hãy xét đoạn chương trình sau

```
int a = 7, b = 2;
```

```
float c = a / b;
```

```
printf(“%d / %d = %2.1f”, a,b,c);
```

Câu hỏi đặt ra là kết quả in ra trên màn hình là gì ?

Câu trả lời là: **7 / 2 = 3.0**

Chắc bạn sẽ rất ngạc nhiên, bởi lẽ ai cũng biết 7 chia 2 phải bằng 3.5 mới đúng !?.

Chuyển kiểu, “Ép kiểu” – *Casting*

Như vậy, để linh hoạt hơn trong những tình huống đặc biệt, ngôn ngữ C (Hoặc những ngôn ngữ lập trình “Tựa C” như Java, C Sharp, ...) cung cấp một kỹ thuật, được gọi là “**Chuyển kiểu**” hay “**Ép kiểu**” đối với dữ liệu hoặc biến trong chương trình, và đoạn chương trình trên sẽ được viết lại như sau

```
int a = 7, b = 2;
```

```
float c = (float) a / b;
```

```
printf(“%d / %d = %2.1f”, a,b,c);
```

Lúc này, kết quả in ra trên màn hình sẽ đúng như bạn muốn

$$7 / 2 = 3.5$$

Cú pháp:

$$\left\{ \begin{array}{l} \text{(kiểu dữ liệu)} \\ \text{(kiểu_dữ_liệu)} \end{array} \right\} \begin{array}{l} \text{<biến_cần_chuyển_kiểu>;} \\ \text{<dữ_liệu_cần_chuyển_kiểu>;} \end{array}$$

Chuyển kiểu, “Ép kiểu” – *Casting*

Trong C, việc ép kiểu được phân ra làm 2 loại

Ép kiểu ngầm định

```
float v = 23;           //--- Tự động chuyển v = 23.0  
int f = 7.23;           //--- Tự động chuyển f = 7
```

Ép kiểu tường minh

```
int a=10, b=4;  
float c = (float) a/b;    //--- Chuyển giá trị của a thành số thực  
                           //--- sau đó thực hiện phép chia a cho b  
                           //--- Lúc này, b tạm thời cũng được  
                           //--- chuyển thành số thực (một cách tạm thời)
```

Thứ tự ưu tiên của phép toán

Thứ tự ưu tiên quyết định trật tự thực hiện các toán tử trên các biểu thức. Bảng dưới đây liệt kê thứ tự thực hiện các toán tử trong Java.

| Toán tử | Mô tả |
|---------|---|
| 1 | Các toán tử đơn như +, -, ++, -- |
| 2 | Các toán tử số học và các toán tử dịch như *, /, +, -, <<, >> |
| 3 | Các toán tử quan hệ như >, <, >=, <=, =, != |
| 4 | Các toán tử logic và Bit như &&, , &, , ^ |
| 5 | Các toán tử gán như =, *=, /=, +=, -= |

Thay đổi thứ tự ưu tiên của các toán tử

- Để thay đổi thứ tự ưu tiên trên một biểu thức, bạn có thể sử dụng dấu ngoặc đơn ():
- Phần được giới hạn trong ngoặc đơn được thực hiện trước.
- Nếu dùng nhiều ngoặc đơn lồng nhau thì toán tử nằm trong ngoặc đơn phía trong sẽ thực thi trước, sau đó đến các vòng phía ngoài.
- Trong phạm vi một cặp ngoặc đơn thì quy tắc thứ tự ưu tiên vẫn giữ nguyên tác dụng.

Ví dụ (1)

```
1 public class Test {
2     public static void main(String[] args) {
3         int a = 20;
4         int b = 5;
5         int c = 10;
6
7         System.out.println("a + b * c    = " + (a + b * c));
8         System.out.println("(a + b) * c = " + ((a + b) * c));
9         System.out.println("a / b - c    = " + (a / b - c));
10        System.out.println("a / (b - c) = " + (a / (b - c)));
11    }
12 }
```

Kết quả:

```
a + b * c    = 70
(a + b) * c = 250
a / b - c    = -6
a / (b - c) = -4
```

Ví dụ (2)

```
class Calculation {
    int z;

    public void addition(int x, int y) {
        z = x + y;
        System.out.println("The sum of the given numbers:"+z);
    }

    public void Subtraction(int x, int y) {
        z = x - y;
        System.out.println("The difference between the given numbers:"+z);
    }
}

public class My_Calculation extends Calculation {
    public void multiplication(int x, int y) {
        z = x * y;
        System.out.println("The product of the given numbers:"+z);
    }

    public static void main(String args[]) {
        int a = 20, b = 10;
        My_Calculation demo = new My_Calculation();
        demo.addition(a, b);
        demo.Subtraction(a, b);
        demo.multiplication(a, b);
    }
}
```

Tổng kết

- Primitive data types: **boolean, char, int, float, ...**
- **Identifier** là gì?, các quy ước liên quan đến identifier. **Camel case** ?.
- Đối tượng Scanner dùng để làm gì ?. Nhớ được bao nhiêu phương thức của Scanner object (**nextInt, nextFloat, nextLine, ...**)
- Có bao nhiêu loại phép toán, độ ưu tiên của phép toán
- Hiểu thế nào về **casting** ?

Tham khảo

- **Primitive Data Types, The Java™ Tutorials, Oracle Document**

(<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html>)

- **Java Scanner Class, Java point**

(<https://www.javatpoint.com/Scanner-class>)

- **Class scanner, Oracle document**

(<https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html>)

- **Formatting**

(<https://docs.oracle.com/javase/tutorial/essential/io/formatting.html>)

THANK YOU VERY MUCH



Q/A: khoihue@gmail.com