



BỘ MÔN CÔNG NGHỆ PHẦN MỀM
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 02. Cú pháp Java cơ bản


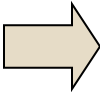
Mục tiêu bài học

- Nêu được các quy ước đặt tên trong các chương trình Java
- Tạo ra các định điều kiện, cấu trúc lặp và rẽ nhánh danh hợp lệ
- Mô tả các kiểu dữ liệu cơ bản trong Java và cách sử dụng
- Các toán tử
- Sử dụng các câu lệnh
- Giải thích về phạm vi của biến
- Khai báo, khởi tạo các biến và mảng trong Java

Nội dung


1. Định danh
2. Các kiểu dữ liệu
3. Toán tử
4. Cấu trúc điều khiển
5. Mạng

Nội dung


- 
- 
1. Định danh
 2. Các kiểu dữ liệu
 3. Toán tử
 4. Cấu trúc điều khiển
 5. Mạng

1. Định danh

- Định danh:
 - Xâu ký tự thể hiện tên các biến, các phương thức, các lớp và nhãn
- Quy định với định danh:
 - Các ký tự có thể là chữ số, chữ cái, '\$' hoặc '_'
 - Tên không được phép:
 - Bắt đầu bởi một chữ số
 - Trùng với từ khóa
 - Phân biệt chữ hoa chữ thường
 - Yourname, yourname, YourName và yourName là 4 định danh khác nhau



```
An_Identifier  
a_2nd_Identifier  
Go2  
$10
```



```
An-Identifier  
2nd_Identifier  
goto  
10$
```

1. Định danh (2)

- Quy ước với định danh (naming convention):
 - Bắt đầu bằng chữ cái
 - Gói (package): tất cả sử dụng chữ thường
 - `theexample`
 - Lớp (Class): viết hoa chữ cái đầu tiên trong các từ ghép lại
 - `TheExample`
 - Phương thức/thuộc tính (method/field): Bắt đầu bằng chữ thường, viết hoa chữ cái đầu tiên trong các từ còn lại
 - `theExample`
 - Hằng (constants): Tất cả viết hoa
 - `THE_EXAMPLE`

1. Định danh (3)

- Literals

`null true false`

- Từ khóa (keyword)

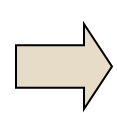
`abstract assert boolean break byte case catch
char class continue default do double else
extends final finally float for if implements
import instanceof int interface long native new
package private protected public return short
static strictfp super switch synchronized this
throw throws transient try void volatile while`

- Từ dành riêng (reserved for future use)

`byvalue cast const future generic goto inner operator
outer rest var volatile`

Nội dung

1. Định danh



2. Các kiểu dữ liệu

3. Toán tử

4. Cấu trúc điều khiển

5. Mảng

2. Các kiểu dữ liệu

- Trong Java kiểu dữ liệu được chia thành hai loại:
 - Kiểu dữ liệu nguyên thủy (primitive)
 - Số nguyên (integer)
 - Số thực (float)
 - Ký tự (char)
 - Giá trị logic (boolean)
 - Kiểu dữ liệu tham chiếu (reference)
 - Mảng (array)
 - Đối tượng (object)

2.1. Kiểu dữ liệu nguyên thủy

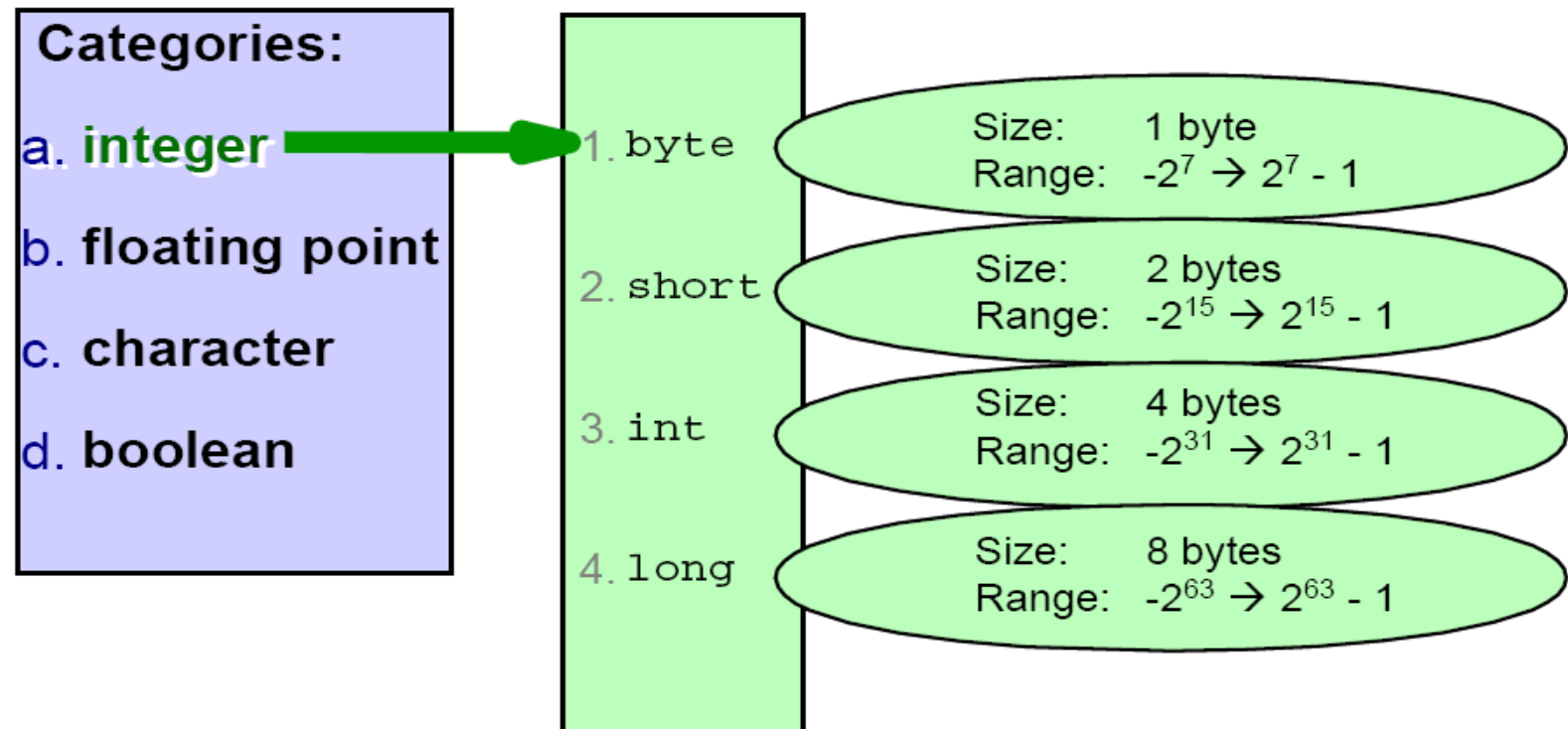
- Mọi biến đều phải khai báo một kiểu dữ liệu
 - Các kiểu dữ liệu cơ bản chứa một giá trị đơn
 - Kích thước và định dạng phải phù hợp với kiểu của nó
- Java phân loại thành 4 kiểu dữ liệu nguyên thủy

Categories:

- a. integer
- b. floating point
- c. character
- d. boolean

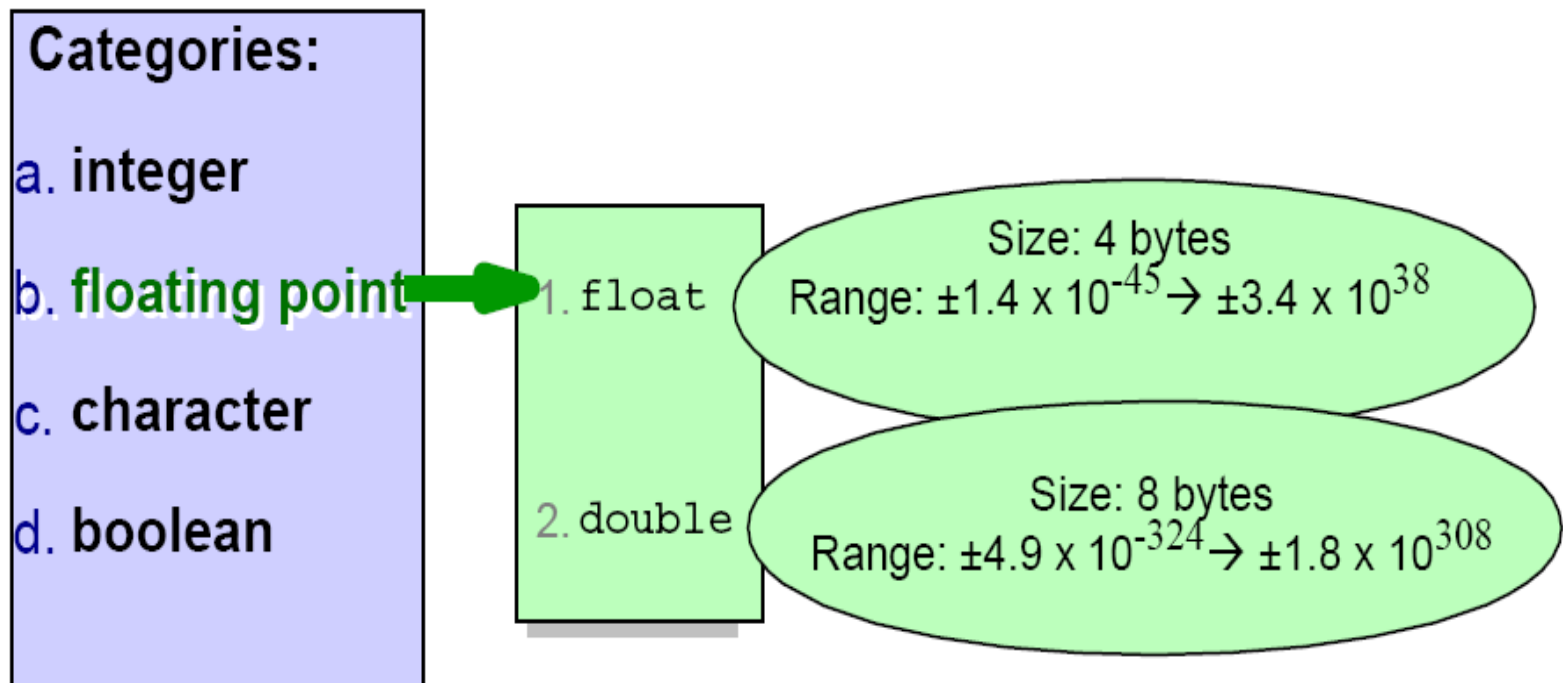
a. Số nguyên

- Số nguyên có dấu
- Khởi tạo với giá trị 0



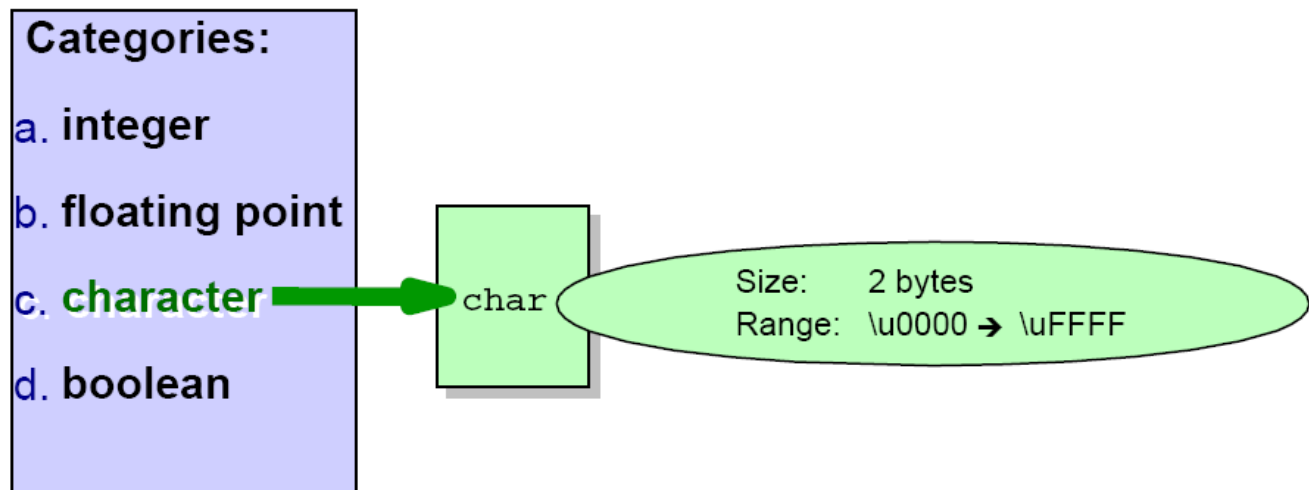
b. Số thực

- Khởi tạo với giá trị 0.0



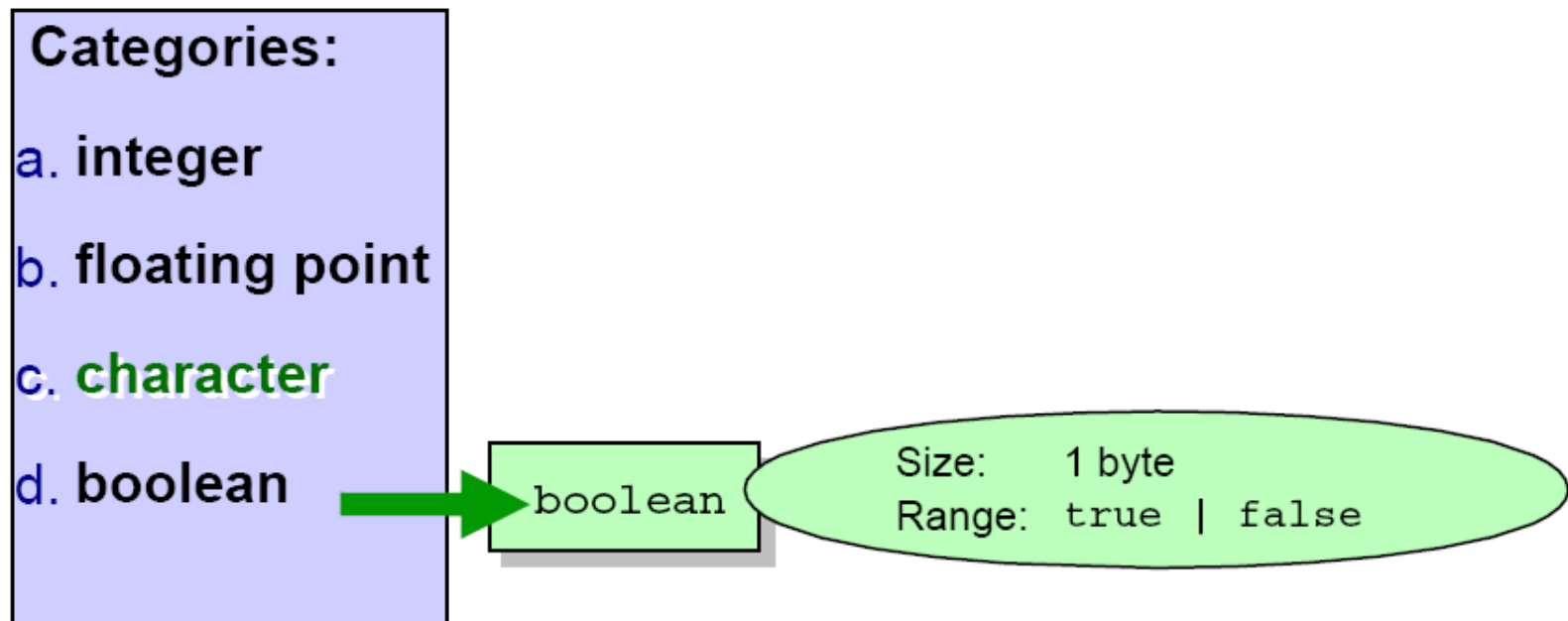
c. Ký tự

- Ký tự Unicode không dấu, được đặt giữa hai dấu nháy đơn
- 2 cách gán giá trị:
 - Sử dụng các chữ số trong hệ 16: `char uni = '\u05D0';`
 - Sử dụng ký tự: `char a = 'A';`
- Giá trị mặc định là giá trị zero (`\u0000`)



d. Giá trị logic

- Giá trị boolean được xác định rõ ràng trong Java
 - Một giá trị int không thể sử dụng thay cho giá trị boolean
 - Có thể lưu trữ giá trị hoặc true hoặc false
- Biến boolean được khởi tạo là false



2.2. Giá trị hằng (literal)

- Literal là một giá trị của các kiểu dữ liệu nguyên thủy và xâu ký tự.
- Gồm 5 loại:
 - integer
 - floating point
 - boolean
 - character
 - string

Literals

integer.....7
floating point...7.0f
boolean.....true
character.....'A'
string....."A"

a. Số nguyên

- Hệ cơ số 8 (Octals) bắt đầu với chữ số 0
 - $032 = 011\ 010(2) = 16 + 8 + 2 = 26(10)$
- Hệ cơ số 16 (Hexadecimals) bắt đầu với 0 và ký tự X
 - $0x1A = 0001\ 1010(2) = 16 + 8 + 2 = 26(10)$
- Kết thúc bởi ký tự “L” thể hiện kiểu dữ liệu long
 - $26L$
- Ký tự hoa, thường cho giá trị bằng nhau
 - $0x1a$, $0x1A$, $0X1a$, $0X1A$ đều có giá trị 26 trong hệ decimal

b. Số thực

- **float** kết thúc bằng ký tự **f** (hoặc **F**)
 - 7.1f
- **double** kết thúc bằng ký tự **d** (hoặc **D**)
 - 7.1D
- **e** (hoặc **E**) được sử dụng trong dạng biểu diễn khoa học:
 - 7.1e2
- Một giá trị thực mà không có ký tự kết thúc đi kèm sẽ có kiểu là **double**
 - 7.1 giống như 7.1d

c. boolean, ký tự và xâu ký tự

- boolean:
 - `true`
 - `False`
- Ký tự:
 - Được đặt giữa 2 dấu nháy đơn
 - Ví dụ: `'a'`, `'A'` hoặc `'\uffff'`
- Xâu ký tự:
 - Được đặt giữa hai dấu nháy kép
 - Ví dụ: `"Hello world"`, `"Xin chào bạn"`,...

d. Escape sequence

- Các ký tự điều khiển nhấn phím
 - `\b` **backspace**
 - `\f` **form feed**
 - `\n` **newline**
 - `\r` **return** (về đầu dòng)
 - `\t` **tab**
- Hiển thị các ký tự đặc biệt trong xâu
 - `\"` **quotation mark**
 - `\'` **apostrophe**
 - `\\` **backslash**

2.3. Chuyển đổi kiểu dữ liệu (Casting)

- Java là ngôn ngữ định kiểu chặt
 - Gán sai kiểu giá trị cho một biến có thể dẫn đến các lỗi biên dịch hoặc các ngoại lệ của JVM
- JVM có thể ngầm định chuyển từ một kiểu dữ liệu hẹp sang một kiểu rộng hơn
- Để chuyển sang một kiểu dữ liệu hẹp hơn, cần phải định kiểu rõ ràng.

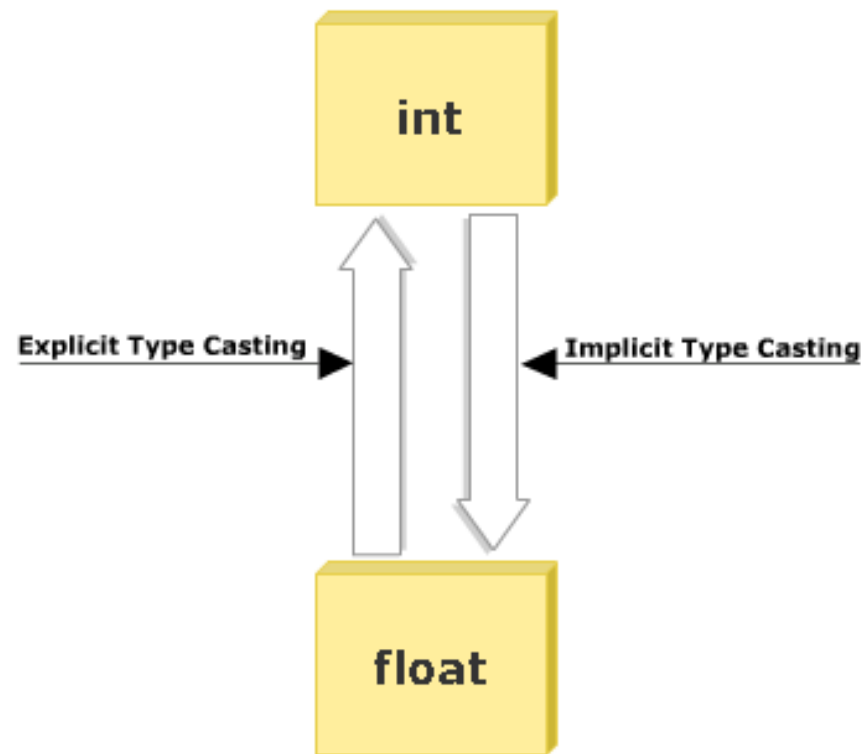
```
int a, b;  
short c;  
a = b + c;
```

```
int d;  
short e;  
e = (short)d;
```

```
double f;  
long g;  
f = g;  
g = f; //error
```

2.3. Chuyển đổi kiểu dữ liệu (2)

- Chuyển đổi kiểu sẽ được thực hiện tự động nếu không xảy ra mất mát thông tin
 - `byte` → `short` → `int` → `long` → `float` → `double`
- Ép kiểu trực tiếp (explicit cast) được yêu cầu nếu có “nguy cơ” giảm độ chính xác



2.4. Khai báo và khởi tạo biến

- Các biến đơn (biến không phải là mảng) cần phải được khởi tạo trước khi sử dụng trong các biểu thức
 - Có thể kết hợp khai báo và khởi tạo cùng một lúc.
 - Sử dụng = để gán (bao gồm cả khởi tạo)
 - Ví dụ:
 - `int i, j;` // Khai báo biến
 - `i = 0;`
 - `int k = i + 1;`
 - `float x = 1.0f, y = 2.0f;`
 - `System.out.println(i);` // In ra 0
 - `System.out.println(k);` // In ra 1
 - `System.out.println(j);` // Lỗi biên dịch

Chú thích

- Java hỗ trợ ba kiểu chú thích như sau:
 - `//` Chú thích trên một dòng
`//` Không xuống dòng
 - `/*` Chú thích một đoạn `*/`
 - `/**` Javadoc `*` chú thích dạng Javadoc `*/`

Câu lệnh

- Các câu lệnh kết thúc bởi dấu;
- Nhiều lệnh có thể viết trên một dòng
- Một câu lệnh có thể viết trên nhiều dòng
 - Ví dụ:

```
System.out.println(  
    "This is part of the same line");
```

```
a=0; b=1; c=2;
```


Nội dung

1. Định danh
2. Các kiểu dữ liệu
- 3. Toán tử
4. Cấu trúc điều khiển
5. Mảng

3. Toán tử (Operators)

- Kết hợp các giá trị đơn hoặc các biểu thức con thành những biểu thức mới, phức tạp hơn và có thể trả về giá trị.
- Java cung cấp nhiều dạng toán tử sau:
 - Toán tử số học
 - Toán tử bit, toán tử quan hệ
 - Toán tử logic
 - Toán tử gán
 - Toán tử một ngôi

✓ $A = B + C$

✓ $Z = Y * Y$

✓ $\text{Result} = (A+B) > (C+D)$



3. Toán tử (2)

- Toán tử số học
 - $+$, $-$, $*$, $/$, $\%$
- Toán tử bit
 - AND: $\&$, OR: $|$, XOR: \wedge , NOT: \sim
 - Dịch bit: \ll , \gg
- Toán tử quan hệ
 - $==$, $!=$, $>$, $<$, $>=$, $<=$
- Toán tử logic
 - $\&\&$, $||$, $!$

3. Toán tử (3)

- Toán tử một ngôi
 - Đảo dấu: +, -
 - Tăng giảm 1 đơn vị: ++, --
 - Phủ định một biểu thức logic: !
- Toán tử gán
 - =, +=, -=, %= tương tự với >>, <<, &, |, ^

Thứ tự ưu tiên của toán tử

- Cho biết toán tử nào thực hiện trước – được xác định bởi các dấu ngoặc đơn hoặc theo ngầm định như sau:
 - Postfix operators `[]` `.` `(params)` `x++` `x--`
 - Unary operators `++x` `--x` `+x` `-x` `~` `!`
 - Creation or cast `new (type)x`
 - Multiplicative `*` `/` `%`
 - Additive `+` `-`
 - Shift `<<` `>>` `>>>` (unsigned shift)
 - Relational `<` `>` `<=` `>=` `instanceof`
 - Equality `==` `!=`
 - Bitwise AND `&`
 - Bitwise exclusive OR `^`
 - Bitwise inclusive OR `|`
 - Logical AND `&&`
 - Logical OR `||`
 - Conditional (ternary) `?:`
 - Assignment `=` `*=` `/=` `%=` `+=` `-=` `>>=` `<<=` `>>>=` `&=` `^=` `|=`

Nội dung

1. Định danh
2. Các kiểu dữ liệu
3. Toán tử
- 4. Cấu trúc điều khiển
5. Mạng

4.1. Lệnh if - else

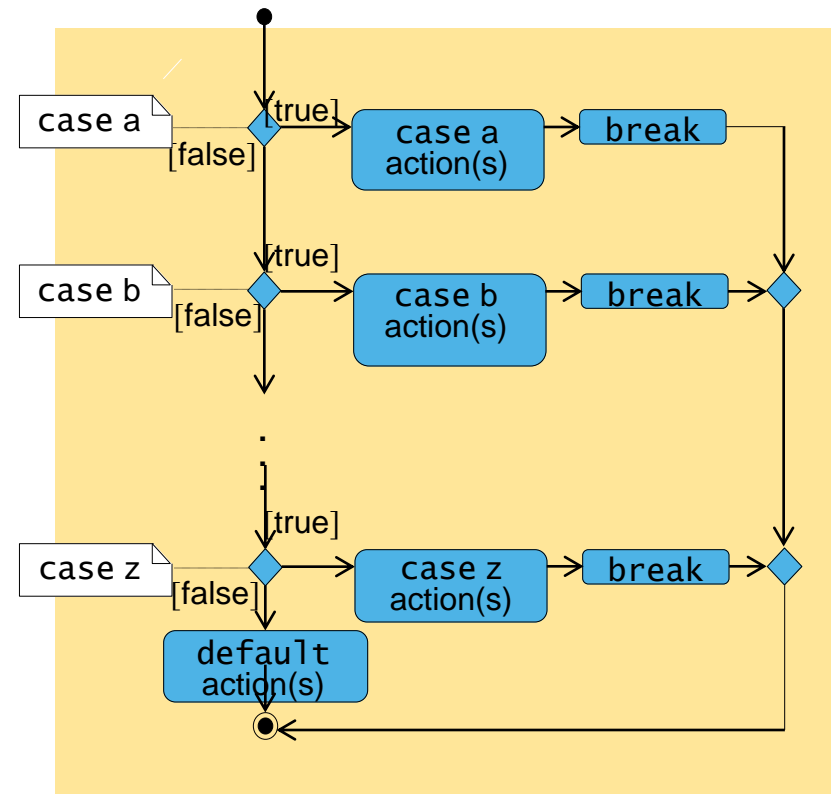
- Cú pháp

```
if (dieu_kien) {  
    cac_cau_lenh;  
}  
else {  
    cac_cau_lenh;  
}
```

- Biểu thức điều kiện nhận giá trị boolean
- Mệnh đề else là tùy chọn

4.2. Lệnh switch - case

- Kiểm tra một biến đơn với nhiều giá trị khác nhau và thực hiện trường hợp tương ứng
 - **break:** Thoát khỏi lệnh switch-case
 - **default** kiểm soát các giá trị nằm ngoài các giá trị case:

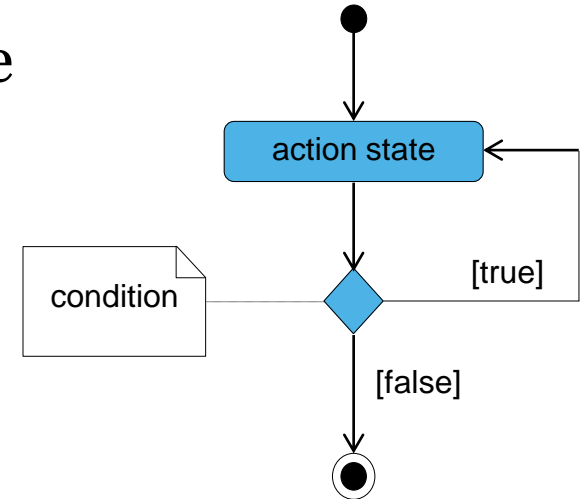


4.3. Vòng lặp while và do while

- Thực hiện một câu lệnh hoặc một khối lệnh khi điều kiện vẫn nhận giá trị true
 - `while()` thực hiện 0 hoặc nhiều lần
 - `do...while()` thực hiện ít nhất một lần

```
int x = 2;  
while (x < 2) {  
    x++;  
    System.out.println(x);  
}
```

```
int x = 2;  
do {  
    x++;  
    System.out.println(x);  
} while (x < 2);
```



4.4. Vòng lặp for

- Cú pháp:

```
for (start_expr; test_expr; increment_expr) {  
    // code to execute repeatedly  
}
```

- 3 biểu thức đều có thể vắng mặt
- Có thể khai báo biến trong câu lệnh for
 - Thường sử dụng để khai báo một biến đếm
 - Thường khai báo trong biểu thức “start”
 - Phạm vi của biến giới hạn trong vòng lặp

- Ví dụ:

```
for (int index = 0; index < 10; index++) {  
    System.out.println(index);  
}
```

4.5. Các lệnh thay đổi cấu trúc điều khiển

- **break**
 - Có thể được sử dụng để thoát ra ngoài câu lệnh switch
 - Kết thúc vòng lặp for, while hoặc do...while
 - Có hai dạng:
 - Gắn nhãn: Tiếp tục thực hiện câu lệnh tiếp theo sau vòng lặp được gắn nhãn
 - Không gắn nhãn: Thực hiện câu lệnh tiếp theo bên ngoài vòng lặp

4.5. Các lệnh thay đổi cấu trúc điều khiển (2)

- **continue**
 - Có thể được sử dụng cho vòng lặp for, while hoặc do...while
 - Bỏ qua các câu lệnh còn lại của vòng lặp hiện thời và chuyển sang thực hiện vòng lặp tiếp theo.

4.6. Phạm vi biến

- Phạm vi của biến là vùng chương trình mà trong đó biến có thể được tham chiếu đến
 - Các biến được khai báo trong một phương thức thì chỉ có thể truy cập trong phương thức đó
 - Các biến được khai báo trong vòng lặp hoặc khối lệnh thì chỉ có thể truy cập trong vòng lặp hoặc khối lệnh đó

```

int a = 1;
for (int b = 0; b < 3; b++) {
    int c = 1;
    for (int d = 0; d < 3; d++) {
        if (c < 3) c++;
    }
    System.out.print(c);
    System.out.println(b);
}

a = c; // ERROR! c is out of scope

```

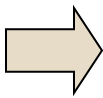
abcd

abc

a

Nội dung

1. Định danh
2. Các kiểu dữ liệu
3. Toán tử
4. Cấu trúc điều khiển



5. Mạng

5. Mảng (array)

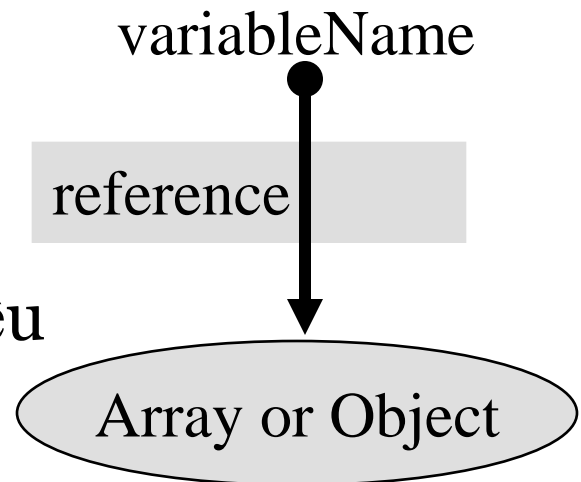
- Tập hợp hữu hạn các phần tử cùng kiểu
- Phải được khai báo trước khi sử dụng
- Khai báo:

- Cú pháp:

- `kiểu_dlieu[] ten_mang = new kiểu_dlieu[KT_MANG];`
- `kiểu_dlieu ten_mang[] = new kiểu_dlieu[KT_MANG];`

- Ví dụ:

- `char c[] = new char[12];`



5.1. Khai báo và khởi tạo mảng

- Khai báo, khởi tạo giá trị ban đầu:
 - **Cú pháp:**
 - `kieu_dl[] ten_mang = {ds_gia_tri_cac_ptu};`
 - **Ví dụ:**
 - `int[] number = {10, 9, 8, 7, 6};`
- Nếu không khởi tạo → nhận giá trị mặc định tùy thuộc vào kiểu dữ liệu.
- Luôn bắt đầu từ phần tử có chỉ số 0

Ví dụ - mảng

Tên của mảng (tất cả các thành phần trong mảng có cùng tên, c)

c.length: cho biết độ dài của mảng c

Chỉ số (truy nhập đến các thành phần của mảng thông qua chỉ số)

c[0]

-45

c[1]

6

c[2]

0

c[3]

72

c[4]

1543

c[5]

-89

c[6]

0

c[7]

62

c[8]

-3

c[9]

1

c[10]

6453

c[11]

78

5.2. Mảng nhiều chiều

- Bảng với các dòng và cột
 - Thường sử dụng mảng hai chiều
 - Ví dụ khai báo mảng hai chiều `b[2][2]`
 - `int b[][] = { { 1, 2 }, { 3, 4 } };`
 - 1 và 2 được khởi tạo cho `b[0][0]` và `b[0][1]`
 - 3 và 4 được khởi tạo cho `b[1][0]` và `b[1][1]`
 - `int b[3][4];`

5.2. Mảng nhiều chiều (2)

	Column 0	Column 1	Column 2	Column 3
Row 0	b[0][0]	b[0][1]	b[0][2]	b[0][3]
Row 1	b[1][0]	b[1][1]	b[1][2]	b[1][3]
Row 2	b[2][0]	b[2][1]	b[2][2]	b[2][3]

Chỉ số cột

Chỉ số hàng

Tên mảng