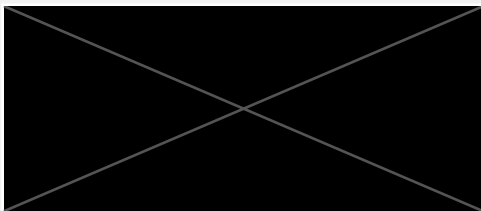


Bài 2

# Java cơ bản



# Nội dung

1. Giới thiệu về Java
2. Định danh
3. Các kiểu dữ liệu
4. Toán tử
5. Cấu trúc điều khiển
6. Mạng



# 1

## Giới thiệu về Java



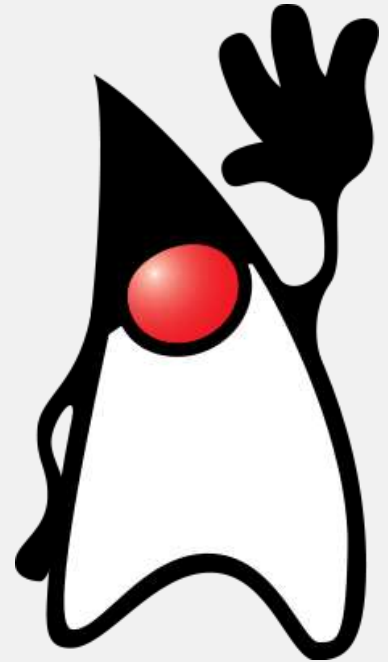
# Ngôn ngữ lập trình Java

- Ngôn ngữ lập trình Java được phát triển vào năm 1991 bởi Sun Microsystems (nay là Oracle)
- Tiêu chí phát triển:  
"Write Once, Run Anywhere"



# Java platform

- Java Platform – nền tảng Java
  - Được xây dựng để phát triển các ứng dụng và phân phối trên môi trường đa nền (các HĐH, điện thoại, thiết bị nhúng, enterprise server...)
  - Sử dụng ngôn ngữ Java (và một số ngôn ngữ khác)
- Tránh nhầm lẫn với ngôn ngữ lập trình Java

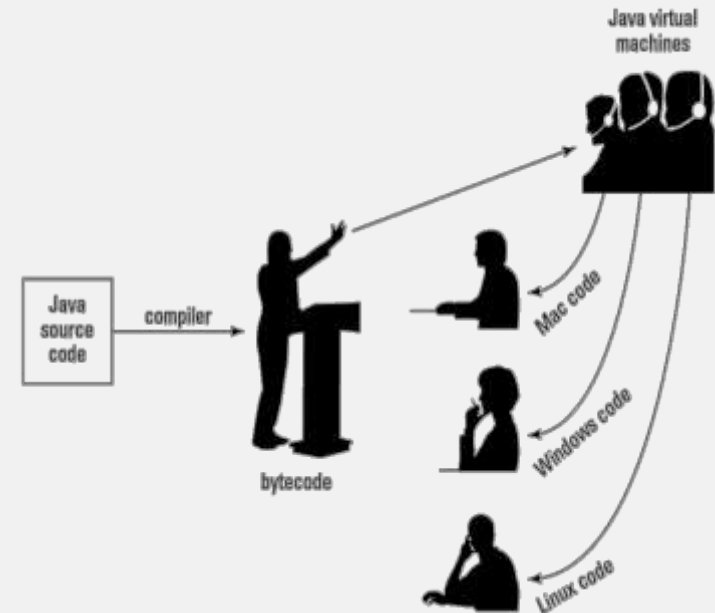


# Java platform

- Các thành phần của Java Platform
  - Các API
    - Java Platform cung cấp các API để lập trình viên không cần phải sử dụng các API của HĐH
  - Java Virtual Machine (JVM)
    - Có thể chạy trên các software platform khác hoặc trực tiếp trên phần cứng
    - Mỗi một platform sử dụng một JVM riêng

# Mô hình biên dịch của Java

- Mô hình biên dịch của Java platform
  - Mã nguồn được biên dịch thành Java byte-code; sau đó được thông dịch trên JVM thành các mã lệnh thực thi bởi trình thông dịch Just-In-Time (JIT)



# Cú pháp cơ bản

- Là ngôn ngữ lập trình phân biệt chữ hoa, chữ thường (case-sensitive)
- Cú pháp tương tự C/C++



# Cài đặt

- Cài Java Development Kit (JDK)
  - <http://www.oracle.com/technetwork/java/javase/downloads>
- Cài IDE
  - Notepad / Notepad++ (<https://notepad-plus-plus.org>)
  - Eclipse (<http://www.eclipse.org>)
  - NetBeans (<http://netbeans.org>)
  - IntelliJ IDEA (<http://www.jetbrains.com/idea>)

# 2

## Định danh

Identifier




# Định danh

- Mỗi đối tượng là duy nhất, dù trạng thái của nó có thể giống các đối tượng khác




# 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$
```

# Quy ước đặt tên

- 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`

# Các từ khóa

- 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`

# 3

## Các kiểu dữ liệu

`integer, float, char, boolean, String...`



# 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)



# Kiểu dữ liệu nguyên thủy

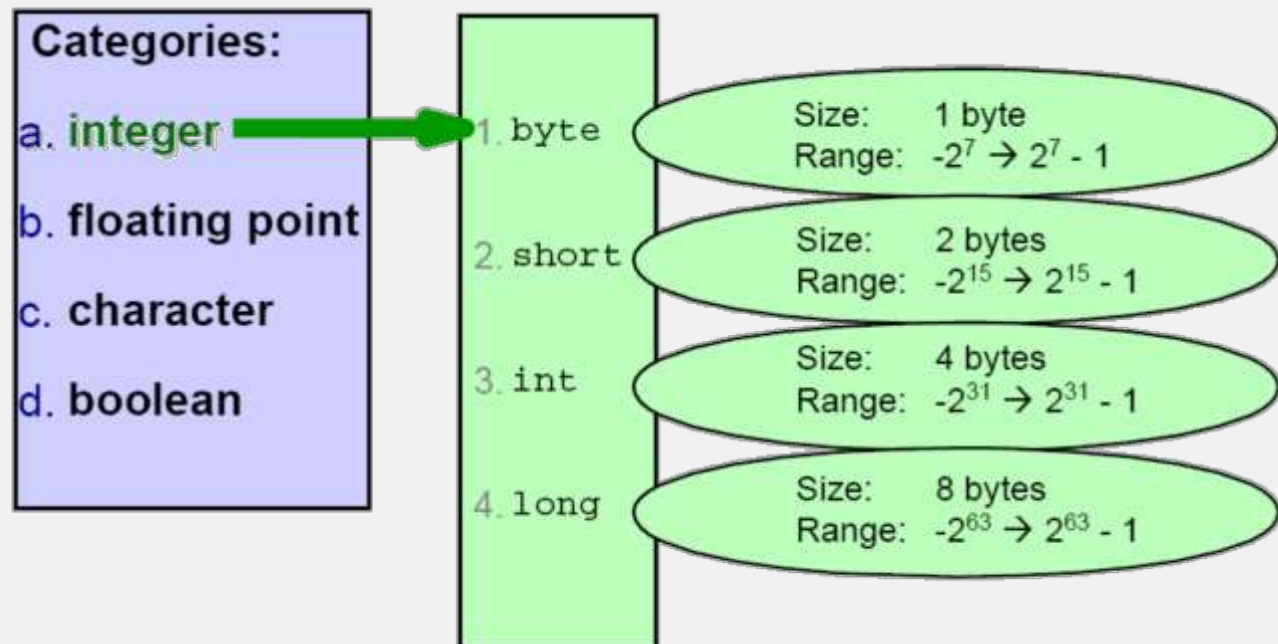
- Mọi biến đều phải khai báo một kiểu dữ liệu
  - Các kiểu dữ liệu cơ bản chứa một giá trị đơn
  - Kích thước và định dạng phải phù hợp với kiểu của nó
- Java phân loại thành 4 kiểu dữ liệu nguyên thủy

## **Categories:**

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

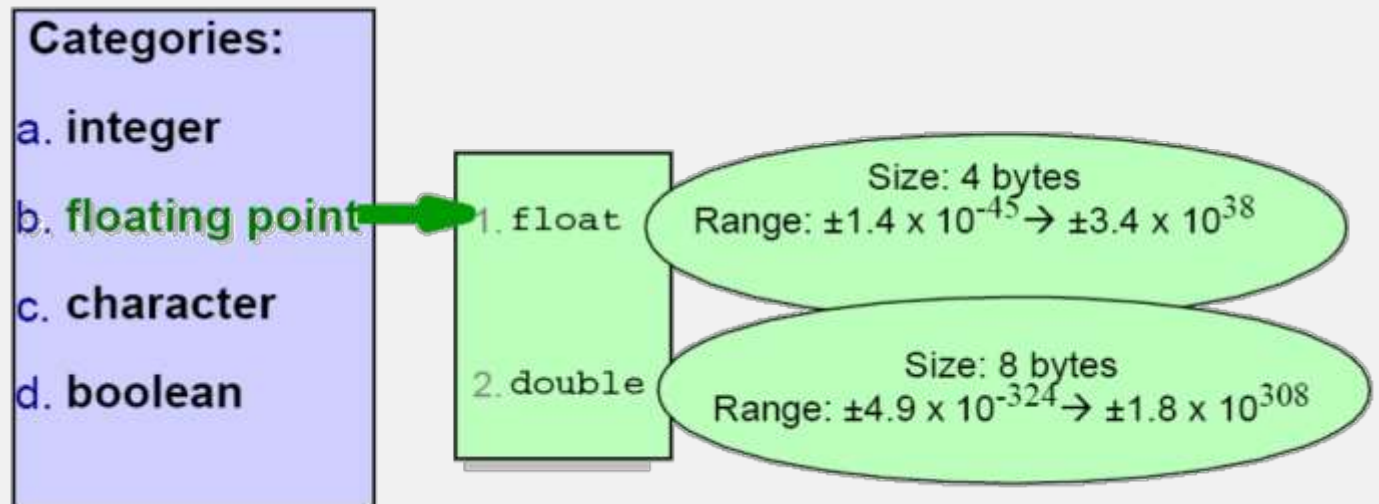
# Số nguyên

- Số nguyên có dấu
  - Giá trị mặc định: 0



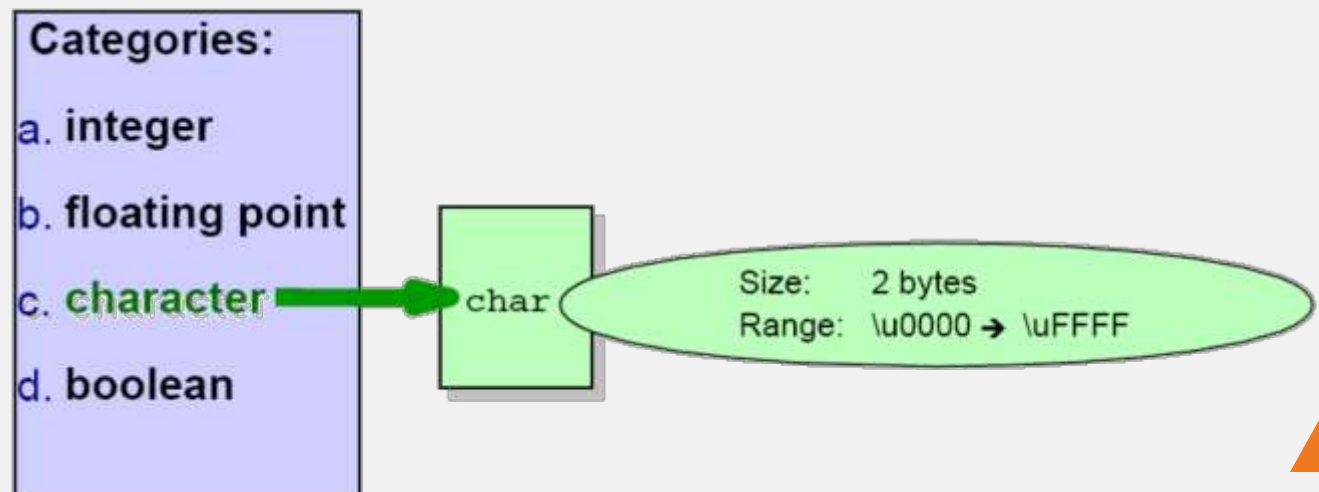
## b. Số thực

- Số thực dấu phẩy động
  - Giá trị mặc định: 0.0



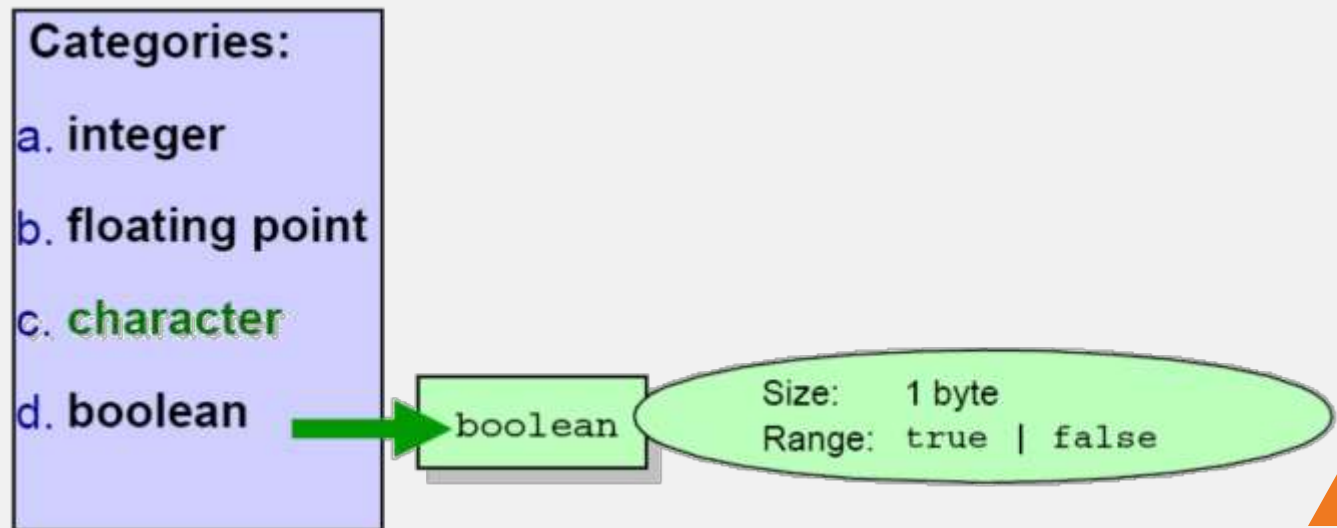
# Ký tự

- 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`)



# Nguyên dạng

- 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



# Nguyên dạng

- Giả sử ta có dòng lệnh

```
int i = 5;
```

Giá trị này ở đâu ra?

- Giá trị này được gọi là *nguyên dạng* hay *giá trị hằng* (*literal*)

# Nguyên dạng

- 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

<u>Literals</u>	
integer.....	7
floating point...	7.0f
boolean.....	true
character.....	'A'
string.....	"A"

# α. 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 chuỗi
  - `\"` **quotation mark**
  - `\'` **apostrophe**
  - `\\` **backslash**

# 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.

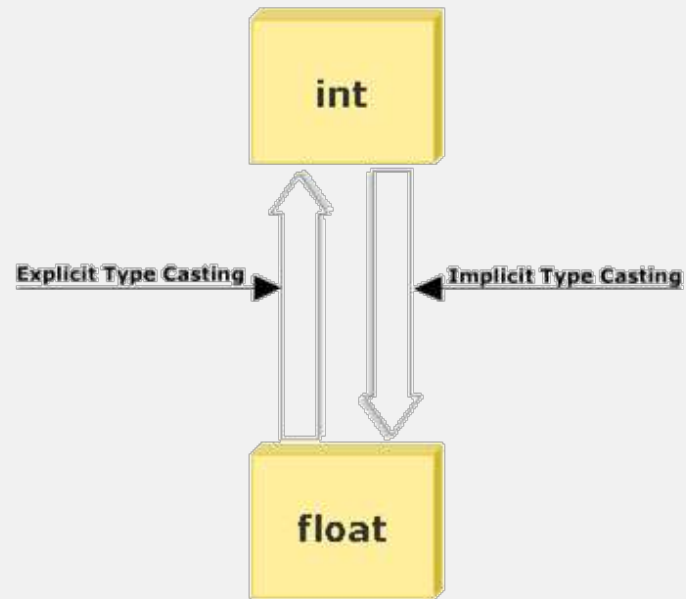
```
int a, b;  
short c;  
a = b + c;
```

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

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

# Chuyển đổi kiểu dữ liệu (casting)

- 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



# Ví dụ - chuyển đổi kiểu

```
long p = (long) 12345.56; // p == 12345
```

```
int g = p;    // không hợp lệ dù kiểu int  
              //có thể lưu giá trị 12345
```

```
char c = 't';
```

```
int j = c;    // tự động chuyển đổi
```

```
short k = c; // không hợp lệ
```

```
short k = (short) c; // ép kiểu trực tiếp
```

```
float f = 12.35; // không hợp lệ
```

# 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 `*/`

```
/**  
 * @param args the command line arguments  
 */
```



# 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;
```

# 4

## Toán tử

Toán tử số học, toán tử logic...



# 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

# Toán tử

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

# Toán tử

- 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

Operators	Precedence
postfix	expr++ expr--
unary	++expr --expr +expr -expr ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= &= ^=  = <<= >>= >>>=

# 5

## Cấu trúc điều khiển

`if-else, switch-case, while...`



# 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

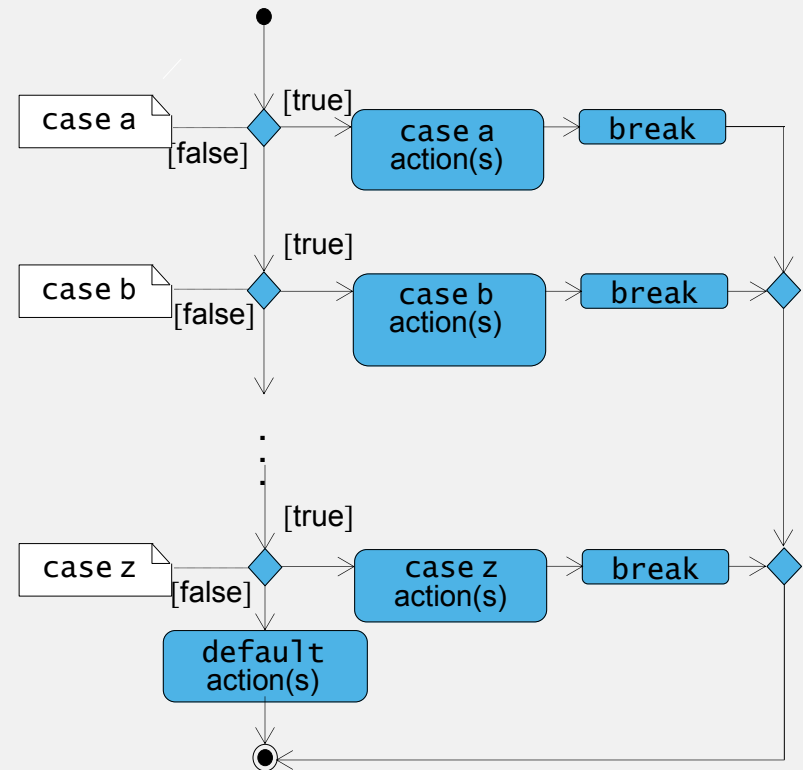


# Ví dụ - Kiểm tra số chẵn – lẻ

```
class CheckNumber
{
    public static void main(String args[])
    {
        int num =10;
        if (num %2 == 0)
            System.out.println (num+ “la so chan”);
        else
            System.out.println (num + “la so le”);
    }
}
```

# 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:



# Ví dụ - Lệnh switch - case

```
switch (day) {  
    case 0:  
    case 1:  
        rule = "weekend";  
        break;  
    case 2:  
    ...  
    case 6:  
        rule = "weekday";  
        break;  
    default:  
        rule = "error";  
}
```

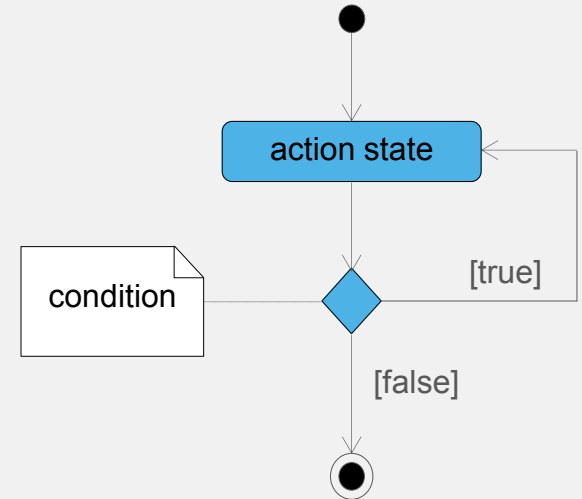
```
if (day == 0 || day == 1) {  
    rule = "weekend";  
} else if (day > 1 && day < 7)  
{  
    rule = "weekday";  
} else {  
    rule = error;  
}
```

# 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);
```



# Ví dụ - Vòng lặp while

```
class WhileDemo{  
    public static void main(String args[]){  
        int a = 5, fact = 1;  
        while (a >= 1){  
            fact *=a;  
            a--;  
        }  
        System.out.println("The Factorial of 5  
                             is "+fact);  
    }  
}
```

# 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);  
}
```

# Ví dụ - vòng lặp for

```
class ForDemo
{
    public static void main(String args[])
    {
        int i=1, sum=0;
        for (i=1;i<=10;i+=2)
            sum+=i;

        System.out.println ("Sum of first five
                             old numbers is " + sum);
    }
}
```

# Vòng lặp for và while

- Các câu lệnh for và while cung cấp chức năng tương đương nhau
- Các cấu trúc lặp thường được sử dụng trong các tình huống khác nhau
  - while được sử dụng cho lặp từ đầu đến cuối
  - for được sử dụng để lặp với số vòng lặp xác định

```
int sum = 0;
int index = 1;
while (index <= 10) {
    sum += index;
    index++;
}
```

```
int sum = 0;
for (int index = 1; index <= 10; index++) {
    sum += index;
}
```



# 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

# Các lệnh thay đổi cấu trúc điều khiển

- 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.

# Ví dụ - break và continue

```
public int myMethod(int x) {  
    int sum = 0;  
    outer: for (int i=0; i<x; i++) {  
        inner: for (int j=i; j<x; j++){  
            sum++;  
            if (j==1) continue;  
            if (j==2) continue outer;  
            if (i==3) break;  
            if (j==4) break outer;  
        }  
    }  
    return sum;  
}
```

```
graph TD
    subgraph Code
        L1[public int myMethod(int x) {]
        L2[    int sum = 0;]
        L3[    outer: for (int i=0; i<x; i++) {]
        L4[        inner: for (int j=i; j<x; j++){]
        L5[            sum++;]
        L6[            if (j==1) continue;]
        L7[            if (j==2) continue outer;]
        L8[            if (i==3) break;]
        L9[            if (j==4) break outer;]
        L10[        }]
        L11[    }]
        L12[    return sum;]
        L13[}]
    end

    L6 --> L4
    L7 --> L3
    L8 --> L11
    L9 --> L11
    L10 --> L3
    L11 --> L12
```

## 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**

# 6

## Mảng

Array



# Mảng (array)

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

```
kieu_dlieu[] ten_mang = new kieu_dlieu[KT_MANG];  
kieu_dlieu ten_mang[] = new kieu_dlieu[KT_MANG];
```
  - Ví dụ:

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

# 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



# Khai báo và khởi tạo mảng

- Ví dụ:

```
int MAX = 5;  
boolean bit[] = new boolean[MAX];  
float[] value = new float[2*3];  
int[] number = {10, 9, 8, 7, 6};  
System.out.println(bit[0]); // prints "false"  
System.out.println(value[3]); // prints "0.0"  
System.out.println(number[1]); // prints "9"
```

# 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];`

# Mảng nhiều chiều

	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

# Thank you!

Any questions?

