

C2

## Cú pháp Java cơ bản

*Số tiết lý thuyết:* **30**

*Số tiết thực hành:* **30**

**Class Room:**

## ➤ Giáo trình chính:

- 1. Đoàn Văn Ban, *Lập trình hướng tượng với Java*, NXB Khoa học Kỹ thuật.

## ➤ Tham khảo:

- 1. C. Thomas Wu. McGraw-Hill Inc, *An Introduction to Object-Oriented Programming with Java*.
- 2. J.N. Patterson Hume and Christine Stephenson, *Introduction to Programming in Java, first edition*.
- 3. Elliot B. Koffman and Paul A.T. Wolfgang. John Wiley & Sons Inc, *Objects, Abstractions, Data Structures and Design using Java*, 2011.

# Mục tiêu

➤ **Mục tiêu:** Trình bày các đặc tính thường thấy ở một ngôn ngữ lập trình bao gồm **biến, mảng, kiểu dữ liệu, toán tử và các cấu trúc điều khiển luồng.**

- Biến.
- Định danh
- Các kiểu dữ liệu
- Toán tử
- Cấu trúc điều khiển luồng
- Mảng

➤ **Biến** là một ***vùng nhớ*** được đặt tên để quản lý trong lập trình.

❖ Các đối tượng được lưu trữ ***trạng thái*** trong các ***thuộc tính***.

❖ Trong ngôn ngữ lập trình **Java**, các khái niệm **thuộc tính** hoặc **biến** đều được sử dụng, nhưng với những lập trình viên mới thì hai khái niệm này nó như là giống nhau.

- Khai báo biến số gồm 4 thành phần:
  - *Kiểu biến (option)*
  - Kiểu dữ liệu của biến
  - Tên biến
  - Giá trị ban đầu của biến (không bắt buộc)

- Cú pháp

**[kind] datatype identifier [=value] [identifier  
[=value]...];**

**static** int numGears = 6;

➤ **Instance Variables:** Biến (Không phải biến tĩnh) khái niệm của loại biến này được định nghĩa là *không phải biến tĩnh*, loại biến này khi định nghĩa thì không cần sử dụng từ khóa static. a.Ten , b.Ten

- ❖ Biến không tĩnh được biết đến chính là **Instance Variables** bởi vì các giá trị của biến là duy nhất cho mỗi thể hiện (instance) của một lớp (mỗi đối tượng của một lớp khác nhau là khác nhau)
- ❖ Ví dụ: Cùng là lớp xe đạp nhưng mỗi đối tượng xe đạp được tạo ra sẽ có tốc độ khác nhau. Biến speed

## ➤ **Class Variables (Static Fields):** Biến tĩnh

- ❖ Một biến bất kỳ được của loại này được định nghĩa với từ khóa ***static*** nói lên rằng trình biên dịch sẽ chỉ lấy một bản copy của biến này để dùng cho tất cả các trường hợp sử dụng biến bất kể biến đó được dùng bao nhiêu lần.
- ❖ Ví dụ: định nghĩa một trường là các bánh răng của một loại xe đạp được định nghĩa bởi từ khóa ***static*** thì kể từ đó, con số này là cố định. Như đoạn code:

```
static int numGears = 6;
```



➤ **Local Variables** Tương tự như một đối tượng được lưu trữ trạng thái trong các thuộc tính (trường), một ***phương thức*** sẽ thường lưu trữ tạm thời các biến cục bộ (**Local Variables**).

❖ Cú pháp khai báo biến cục bộ tương tự như khai báo các biến khác. Ví dụ:

```
int count = 0;
```

❖ Biến cục bộ không được truy cập từ ngoài lớp hoặc phương thức nó khai báo.



# Biến

➤ **Parameters: Biến tham số.** Chúng ta xem xét ví dụ về biến tham số, cả trong lớp Bicycle và trong hàm main cho ứng dụng "Hello world", hàm main được viết như sau:

```
public static void main(String[] args)
```

❖ Ở đây biến args là một biến tham số. Điều quan trọng cần nhớ đó là biến tham số luôn luôn là biến và không phải là thuộc tính.

➤ **Biến** là **vùng nhớ** được *đặt tên* có kích thước bằng với kích thước của *kiểu dữ liệu*.

# Định danh

- Tất cả các ngôn ngữ lập trình đều có đặt ra các **quy tắc** và **quy ước** cho việc **đặt tên**.
- Ngôn ngữ lập trình Java cũng không ngoại lệ. Các quy tắc và quy ước cho việc đặt tên biến có thể tóm lại như sau:
  - ❖ Tên phân biệt *chữ hoa* và *chữ thường*.
  - ❖ Một biến có thể được tạo thành từ các chữ cái unicode, dấu đô la '\$' và dấu gạch dưới '\_' tuy nhiên không được bắt đầu bằng dấu đô la '\$' hoặc gạch dưới '\_' và không giới hạn độ dài.

# Định danh

- ❖ Thêm nữa, Java đã quy định dấu đô la không được sử dụng trong tên biến. Bạn có thể tìm được một vài tình huống sử dụng tên tự động sẽ bao gồm dấu đô la, nhưng tên biến thì nên tránh.
- ❖ Tương tự quy ước đã tồn tại ký tự gạch dưới được bắt đầu tên biến, nhưng khuyên không nên sử dụng. Khoảng trắng trong tên biến là không được phép.
- ❖ Khi chọn tên biến, nên sử dụng cả từ, tránh các từ viết tắt khó hiểu. Ví dụ: *cadence*, *speed*, *gear*;

# Định danh

- ❖ Nếu tên được chọn bao gồm một từ, có thể sử dụng toàn bộ đánh vần của từ đó, nếu có từ 2 từ trở lên, hãy viết hoa chữ cái đầu tiên của các từ tiếp theo.
- ❖ Ví dụ: *gearRatio*, *currentGear*, nếu biến chứa giá trị là các hằng số thì có thể viết hoa toàn bộ biến đó.

# Cấu trúc một chương trình Java

- Xác lập thông tin môi trường
- Các thành phần (Tokens):
  - Định danh
  - Từ khóa / từ dự phòng
  - Ký tự phân cách
  - Nguyên dạng (Literals)
  - Toán tử

# Kiểu dữ liệu

- Kiểu dữ liệu cơ sở (Primitive Data Types +String)
- Kiểu dữ liệu tham chiếu (Reference data types)

# Kiểu dữ liệu nguyên thủy

- byte
- char
- boolean
- short
- int
- long
- float
- double



# Tám kiểu dữ liệu nguyên thủy



TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\', '\n', ' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

# Kiểu dữ liệu nguyên thủy

- Không cần khởi gán các giá trị khi định nghĩa các biến. Các biến (fields) được định nghĩa mà không khởi tạo sẽ nhận các giá trị mặc định khi biên dịch.
- `int x; boolean b; long y;`

# Kiểu dữ liệu cơ sở

Kiểu dữ liệu	Giá trị mặc định (cho biến)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

# Kiểu dữ liệu nguyên thủy

➤ Nguyên dạng (Literals): Một literal là một mã nguồn được cố định giá trị.

```
boolean result = true;  
char capitalC = 'C';  
byte b = 100;  
short s = 10000;  
int i = 100000;
```

# Kiểu dữ liệu nguyên thủy

## ➤ Literal dạng số (Integer Literals)

```
// Số 26 kiểu thập phân  
int decVal = 26;  
// Số 26 kiểu hexa  
int hexVal = 0x1a;  
// Số 26 kiểu nhị phân  
int binVal = 0b11010;
```

## ➤ Các Literal dạng số thực (Floating-Point Literals)

```
double d1 = 123.4;  
// tương tự giá trị d1, nhưng trong thẻ hiện số khoa học.  
double d2 = 1.234e2;  
float f1 = 123.4f;
```

# Kiểu dữ liệu nguyên thủy

## ➤ Các Literal dạng chuỗi và ký tự.

- ❖ Ngôn ngữ lập trình Java cũng hỗ trợ một vài ký tự đặc biệt cho char và String \b (backspace), \t (tab), \n (line feed), \f (form feed), \r (carriage return), \" (double quote), \' (single quote), and \\ (backslash).
- ❖ Sử dụng các ký tự gạch dưới cho các Literal dạng số.

```
long creditCardNumber = 1234_5678_9012_3456L;  
long socialSecurityNumber = 999_99_9999L;  
float pi = 3.14_15L;  
long hexBytes = 0xFF_EC_DE_5E;  
long hexWords = 0xCAFE_BABE;  
long maxLong = 0x7fff_ffff_ffff_ffffL;  
byte nybbles = 0b0010_0101;  
long bytes = 0b11010010_01101001_10010100_10010010;
```

# Kiểu dữ liệu nguyên thủy

- ❖ Bạn có thể thay thế các dấu gạch dưới giữa các số, nhưng bạn không thể thay thế dấu gạch dưới đó bằng các khoảng trắng:
  - Ở vị trí bắt đầu và vị trí kết thúc của số.
  - Liên kề dấu chấm của số thập phân trong Literal dạng số thực.
  - Ngay trước chữ F hoặc L.

# Kiểu dữ liệu tham chiếu

- Mảng (Array)
- Lớp (Class)
- Interface

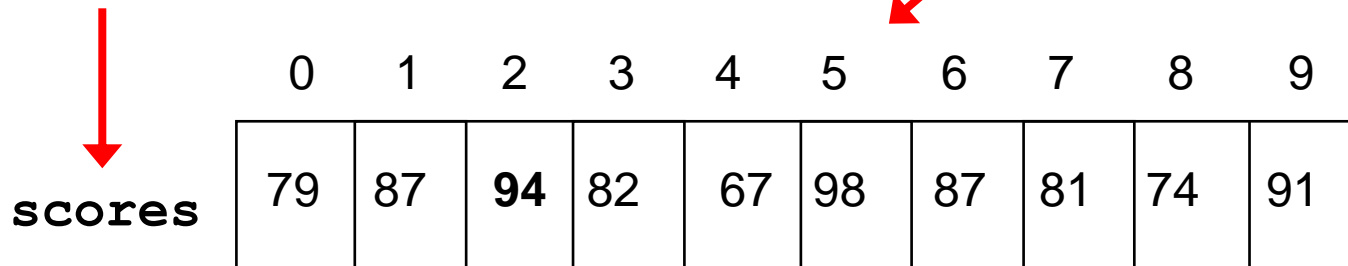


# Mảng Arrays

- Mảng là 1 danh sách các giá trị

Tên mảng

Mỗi phần tử mảng có một chỉ số



	0	1	2	3	4	5	6	7	8	9
<b>scores</b>	79	87	<b>94</b>	82	67	98	87	81	74	91

Mảng có kích thước N thì chỉ số từ không to N-1

Mảng có 10 phần tử thì chỉ số từ 0 to 9

# Arrays

- Mỗi phần tử mảng được truy cập thông qua tên mảng và chỉ số.
- Ví dụ, phần tử:

`scores[2]`

cho giá trị 94 (là phần tử thứ 3 của mảng)

- Mỗi phần tử của mảng được lưu trữ là 1 số nguyên, có kích thước nhất định.

# Arrays

- Các phần tử của mảng có thể được gán giá trị, tính toán, hiển thị:

```
scores[2] = 89;
```

```
scores[first] = scores[first] + 2;
```

```
mean = (scores[0] + scores[1]) / 2;
```

```
System.out.println ("Top = " +  
scores[5]);
```

# Arrays

- Các giá trị được lưu trữ trong mảng được gọi là các phần tử mảng.
- Một mảng lưu trữ nhiều giá trị có **cùng kiểu**.
- Kiểu phần tử mảng có thể là kiểu dữ liệu nguyên thủy hoặc **kiểu tham chiếu**.
- Do vậy ta có thể tạo ra mảng các số nguyên hoặc mảng các chuỗi, Hoặc một mảng có phần tử là kiểu String, etc.
- Trong Java, bản thân mảng cũng là **đối tượng**.
- Do vậy tên của mảng cũng là một biến tham chiếu đối tượng, và mảng phải được khởi tạo.

# Declaring Arrays

- Phạm vi của mảng có thể được định nghĩa:

```
int[] scores = new int[10];
```

- Kiểu dữ liệu của `scores` là `int[]` (một mảng số nguyên)
- Chú ý rằng kiểu dữ liệu của mảng không định nghĩa kích thước của nó, nhưng mỗi đối tượng kiểu đó có một kích thước cụ thể.
- Biến tham chiếu `scores` được thiết lập một mảng mới có 10 phần tử.

# Declaring Arrays

- Một vài ví dụ về khai báo mảng:

```
float[] prices = new float[500];
```

```
boolean[] flags;
```

```
flags = new boolean[20];
```

```
char[] codes = new char[1750];
```

# Bounds Checking

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in bounds (0 to N-1)
- The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called *automatic bounds checking*

# Bounds Checking

- Ví dụ nếu mảng `codes` có 100 phần tử, nó có thể sử dụng các chỉ số từ 0 đến 99.
- Nếu `count` có giá trị 100, thì câu lệnh sau sẽ báo lỗi.:

```
System.out.println (codes[count]);
```

- Một lỗi khá phổ biến khi sử dụng mảng.

problem

```
for (int index=0; index <= 100; index++)  
codes[index] = index*50 + epsilon;
```



# Bounds Checking

- Mỗi một đối tượng mảng có một hằng số `length` lưu trữ kích thước mảng.
- Nó được sử dụng tham chiếu thông qua tên mảng:

`scores.length`

- Chú ý rằng `length` là số phần tử mảng, chứ không phải là chỉ số mảng lớn nhất.

# Alternate Array Syntax

- Dấu ngoặc vuông có thể liên kết với tên mảng hoặc kiểu mảng
- Do đó 2 cách khai báo sau là tương đương:

```
float[] prices;
```

```
float prices[];
```

- Nhưng cách thứ nhất phổ biến và dễ đọc hơn.

# Initializer Lists

- Danh sách giá khởi tạo có thể được dùng để khởi tạo và được khởi tạo trong một bước.
- Các giá trị trong danh sách được cách nhau bởi dấu phẩy và đặt trong dấu ngoặc nhọn.
- Ví dụ:

```
int[] units = {147, 323, 89, 933, 540,  
              269, 97, 114, 298, 476};
```

```
char[] letterGrades = {'A', 'B', 'C', 'D',  
                       'F'};
```

# Initializer Lists

- Chú ý khi một danh sách khởi tạo được sử dụng:
  - ⇒ Toán tử `new` không được sử dụng
  - ⇒ Không có giá trị kích thước cụ thể.
- Kích thước của mảng được quyết định bởi số mục có trong danh sách khởi tạo.
- Một danh sách khởi tạo chỉ được sử dụng khi khai báo mảng.

# Arrays as Parameters

- Mỗi mảng có thể được truyền vào như một tham số cho phương thức.
- Giống như bất kỳ một đối tượng khác, khi tham chiếu một mảng được truyền vào, tạo ra một hình thức và tham chiếu thực sự giống như đối tượng khác.
- Một phần tử của mảng cũng có thể được truyền vào tham số của phương thức.

# Arrays of Objects

- Các phần tử của mảng có thể là đối tượng tham chiếu.
- Phần sau khai báo 25 tham chiếu lưu trữ kiểu `String`

```
String[] words = new String[25];
```

- Nó không tự tạo ra đối tượng `String`
- Được lưu trữ trong một mảng phải được khởi tạo.

# Sao chép mảng

➤ Trong lớp **System**, có một phương thức copy mảng ***arraycopy*** bạn có thể sử dụng để sao chép dữ liệu từ mảng này sang mảng khác.

```
public static void arraycopy(Object src, int srcPos,  
                             Object dest, int destPos, int length)
```

# Sao chép mảng

## ➤ Demo **ArrayCopyDemo**

```
class ArrayCopyDemo {  
    public static void main(String[] args) {  
        String[] copyFrom = {  
            "Affogato", "Americano", "Cappuccino", "Corretto", "Cortado",  
            "Doppio", "Espresso", "Frappucino", "Freddo", "Lungo", "Macchiato",  
            "Marocchino", "Ristretto" };  
  
        String[] copyTo = new String[7];  
        System.arraycopy(copyFrom, 2, copyTo, 0, 7);  
        for (String coffee : copyTo) {  
            System.out.print(coffee + " ");  
        }  
    }  
}
```



# Ép kiểu (Type Casting)

- Kiểu dữ liệu này được chuyển đổi sang một kiểu dữ liệu khác
- Ví dụ

```
float c = 34.89675f;
```

```
int b = (int)c + 10;
```

# Những từ khóa của Java



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

- Các loại toán tử:
  - Toán tử số học (Arithmetic operators)
  - Toán tử dạng Bit (Bitwise operators)
  - Toán tử so sánh (Relational operators)
  - Toán tử logic (Logical operators)
  - Toán tử điều kiện (Conditional operator)
  - Toán tử gán (Assignment operator)

# Các 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	= += -= *= /= %= &= ^=  = <<= >>= >>>=

# Toán tử số học: Arithmetic Operators

- + Addition (Phép cộng)
- Subtraction (Phép trừ)
- \* Multiplication (Phép nhân)
- / Division (Phép chia)
- % Modulus (Lấy số dư)
- ++ Increment (Tăng dần)
- Decrement (Giảm dần)

# Toán tử số học: Arithmetic Operators

`+=`                      Phép cộng và gán

`-=`                      Phép trừ và gán

`*=`                      Phép nhân và gán

`/=`                      Phép chia và gán

`%=`                      Phép lấy số dư và gán

Demo: `ArithmeticDemo`

# Toán tử một ngôi

## ➤ Toán tử một ngôi

Operator	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

➤ Demo: **UnaryDemo, PrePostDemo**

# Toán tử Bit: (Bitwise Operators)

- ~ Phủ định (NOT)
- & Và (AND)
- | Hoặc (OR)
- ^ Exclusive OR
- >> Dịch sang phải (Shift right)
- << Dịch sang trái (Shift left)



# Toán tử so sánh: (Relational Operators)

Các toán tử bình đẳng, quan hệ và điều kiện

= So sánh bằng

!= So sánh khác

< Nhỏ hơn

> Lớn hơn

<= Nhỏ hơn hoặc bằng

>= Lớn hơn hoặc bằng

Demo: **ComparisonDemo,**

# Toán tử Logic: (Logical Operators )

&&                      Logical AND

||                      Logical OR

!                      Logical unary NOT

**ConditionalDemo1, ConditionalDemo2,**

**InstanceOfDemo**

# Toán tử điều kiện (Conditional Operator)

- Cú pháp

**Biểu thức 1 ? Biểu thức 2 : Biểu thức 3;**

- **Biểu thức 1**

Điều kiện kiểu Boolean trả về giá trị True hoặc False

- **Biểu thức 2**

Trả về giá trị nếu kết quả của mệnh đề 1 là True

- **Biểu thức 3**

Trả về giá trị nếu kết quả của mệnh đề 1 là False

# Toán tử gán (Assignment Operator)

= Assignment (Phép gán)

Giá trị có thể được gán cho nhiều biến số

- Ví dụ

**a = b = c = d = 90;**

# Thứ tự ưu tiên của các toán tử

Thứ tự	Toán tử
1.	trong ngoặc tính trước
2.	Các toán tử đơn như $+$ , $-$ , $++$ , $--$
3.	Các toán tử số học và các toán tử dịch như $*$ , $/$ , $+$ , $-$ , $<<$ , $>>$
4.	Các toán tử quan hệ như $>$ , $<$ , $>=$ , $<=$ , $=$ , $!=$
5.	Các toán tử logic và Bit như $\&\&$ , $\ \ $ , $\&$ , $\ $ , $\wedge$
5.	Các toán tử gán như $=$ , $*=$ , $/=$ , $+=$ , $-=$

- Thứ tự của các toán tử có thể được thay đổi bằng cách sử dụng các dấu ngoặc đơn trong mệnh đề

# Các kí tự định dạng xuất dữ liệu (Escape Sequences)

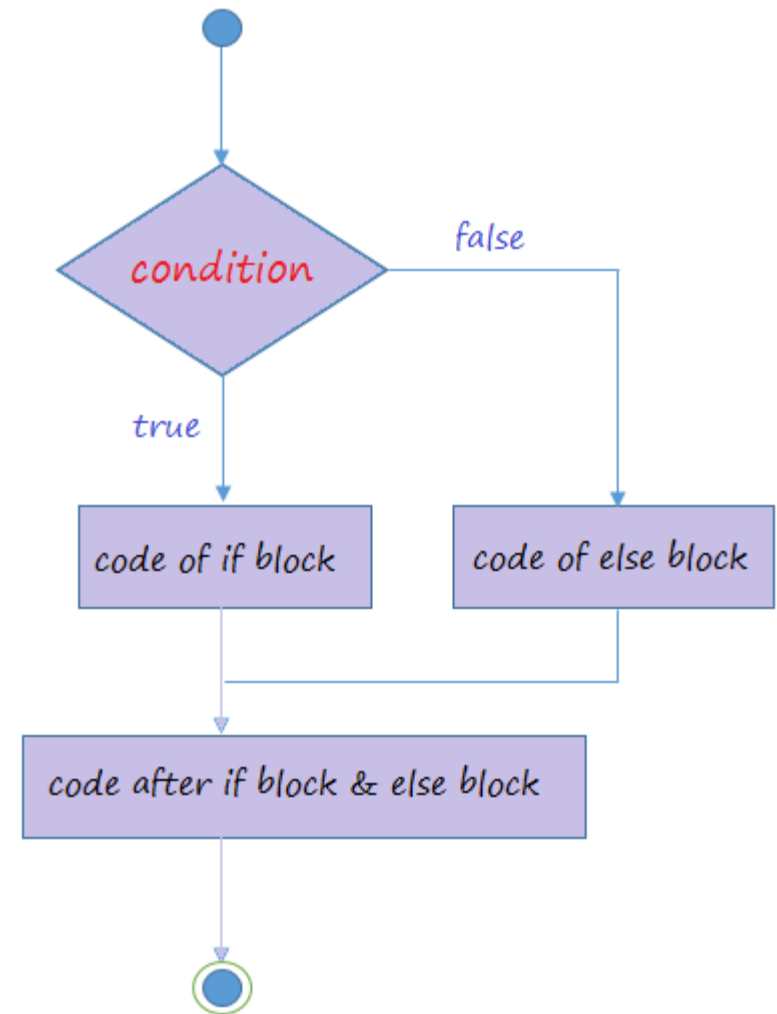
Escape Sequence	Mô tả
<code>\n</code>	Xuống dòng mới
<code>\r</code>	Chuyển con trỏ đến đầu dòng hiện hành
<code>\t</code>	Chuyển con trỏ đến vị trí dừng Tab kế tiếp (ký tự Tab)
<code>\\</code>	In dấu \
<code>\'</code>	In dấu nháy đơn (')
<code>\''</code>	In dấu nháy kép (")

# Các lệnh điều khiển

- Điều khiển rẽ nhánh:
  - Mệnh đề **if-else**
  - Mệnh đề **switch-case**
- Vòng lặp (Loops):
  - Vòng lặp **while**
  - Vòng lặp **do-while**
  - Vòng lặp **for**

# Lệnh if-else

- Cú pháp  
**if (condition)**  
{  
    **action1 statements;**  
}  
**else**  
{  
    **action2 statements;**  
}





# Câu lệnh if-else

- Câu lệnh if-then là câu lệnh cơ bản của tất cả các câu lệnh điều khiển.
- Nó cho chương trình của bạn biết thực thi phần mã nguồn nào chỉ với từ if, kiểm tra các đánh giá là true.
- Ví dụ trong lớp Bicycle phanh giảm bớt tốc độ của xe khi nó đang chuyển động chỉ với từ if. Ví dụ thực thi bằng phương thức `applyBrakes`.



# Bài tập Câu lệnh if-else

- Bài 1. Hãy tạo và gán 2 biến kiểu nguyên. Hiển thị tổng, hiệu, tích, thương của 2 số đó.
- Bài 2: Tạo mảng có 3 phần tử tương ứng với ngày tháng, năm sinh. Hiển thị câu chào, và giới thiệu bản thân và ngày, tháng năm sinh theo mảng.
- Bài 3. Viết chương trình giải phương trình bậc nhất 1 ẩn, biết rằng tạo và gán hai biến số thực a, b.  
 $a \cdot x + b = 0$ ;
- Bài 4. Giải phương trình bậc 2, a,b,c được gán sẵn.

# Bài tập

- `int i = 10;`
- `int n = i++%5;`
- **result** = `someCondition ? value1 : value2;`

# Lệnh switch-case

- Cú pháp

**switch** (expression)

{

**case** 'value1': action1 statement(s);  
                    **break**;

**case** 'value2': action2 statement(s);  
                    **break**;

    :

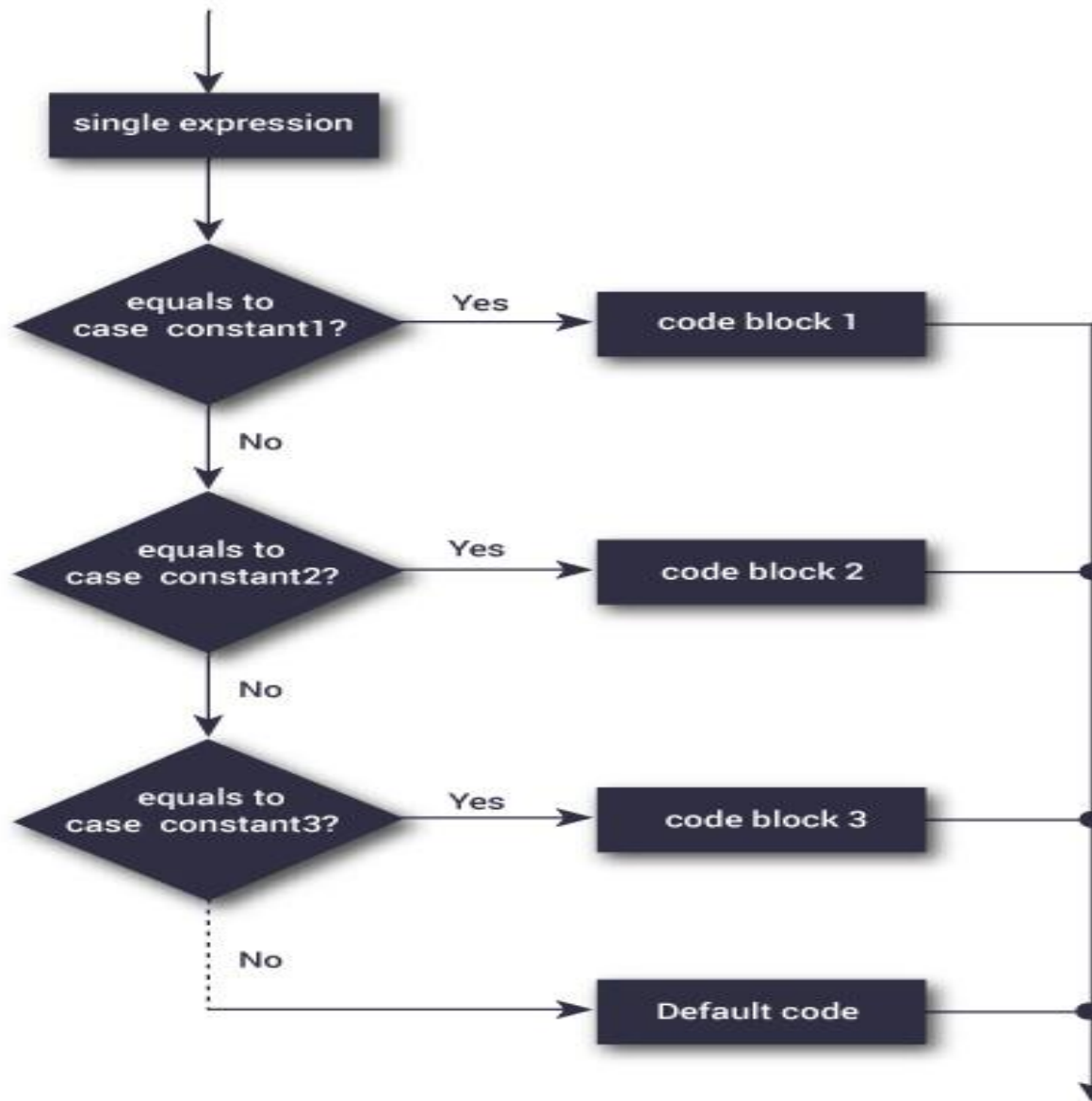
    :

**case** 'valueN': actionN statement(s);  
                    **break**;

**default**: default\_action statement(s);

}

# Lệnh switch-case





# Lệnh switch-case

- Không giống các câu lệnh **if-then** và **if-then-else**, câu lệnh switch có thể thực thi số lượng đếm được các nhánh.
- Một **switch** làm việc với *byte*, *short*, *char*, *int* các kiểu dữ liệu nguyên thủy.
- Nó cũng làm việc với các kiểu **dữ liệu liệt kê** (sẽ được bàn trong kiểu **Enum**), lớp **String**, và một vài lớp đặc biệt khác tất nhiên các dữ liệu nguyên thủy: **Character**, **Byte**, **Short** và **Integer** (Được bàn trong các số và các chuỗi.)

# Lệnh switch-case

- Về kỹ thuật thì **break** ở cuối không yêu cầu bởi vì các câu lệnh sẽ được thực hiện trong lệnh **switch**.
- Sử dụng một **break** được khuyến khích làm thay đổi code, dễ đọc và ít bị lỗi hơn.
- Phần **default** được dùng khi tất cả các phần case khác đều không thỏa mãn.
- Ví dụ: **SwitchDemo2**

# Lệnh switch-case

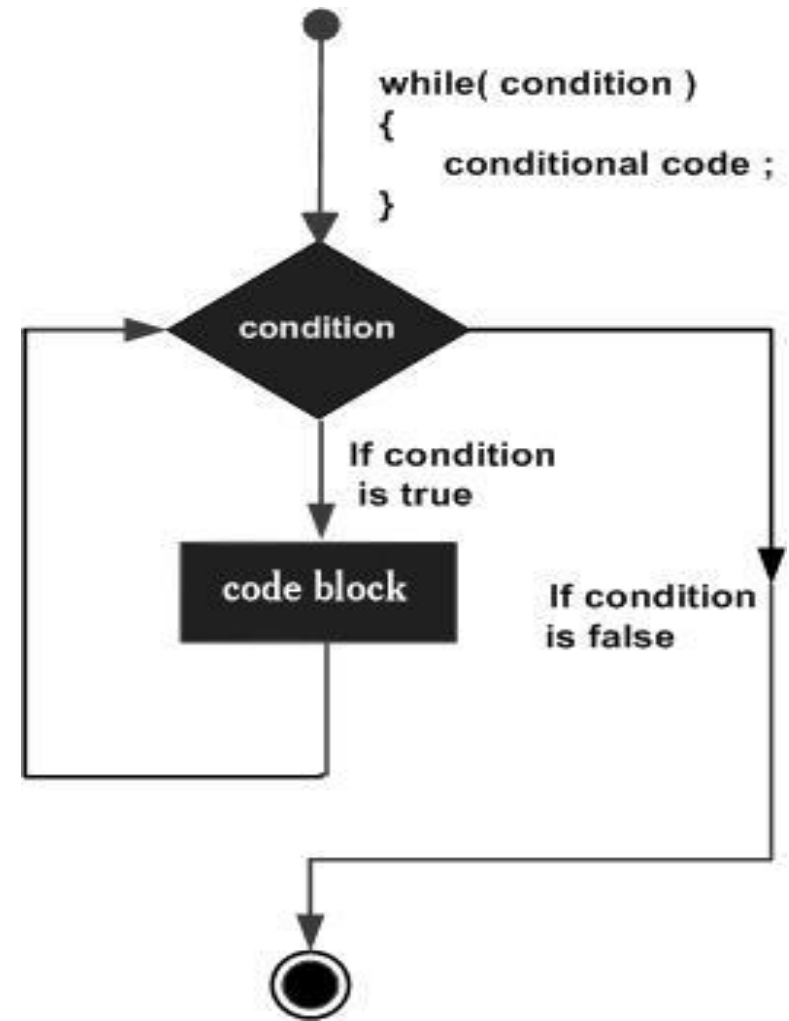
- Từ Java SE 7 trở đi, bạn có thể sử dụng một đối tượng của lớp String trong biểu thức điều kiện của câu lệnh switch.
- Trong ví dụ **StringSwitchDemo** hiển thị số tháng dựa vào giá trị tên của lớp String là month



# Câu lệnh while và do-while

```
while  
(expression)  
{  
    statement(s)  
}
```

Câu lệnh while tính toán biểu thức, biểu thức này phải trả về một giá trị kiểu **boolean**. Nếu biểu thức tính toán ra giá trị true, câu lệnh while thực thi các câu lệnh nằm trong khối lệnh while. Ví dụ **WhileDemo**.



# Câu lệnh **while** và **do-while**

➤ Bạn có thể thực thi một vòng lặp vô hạn sử dụng câu lệnh **while** như sau:

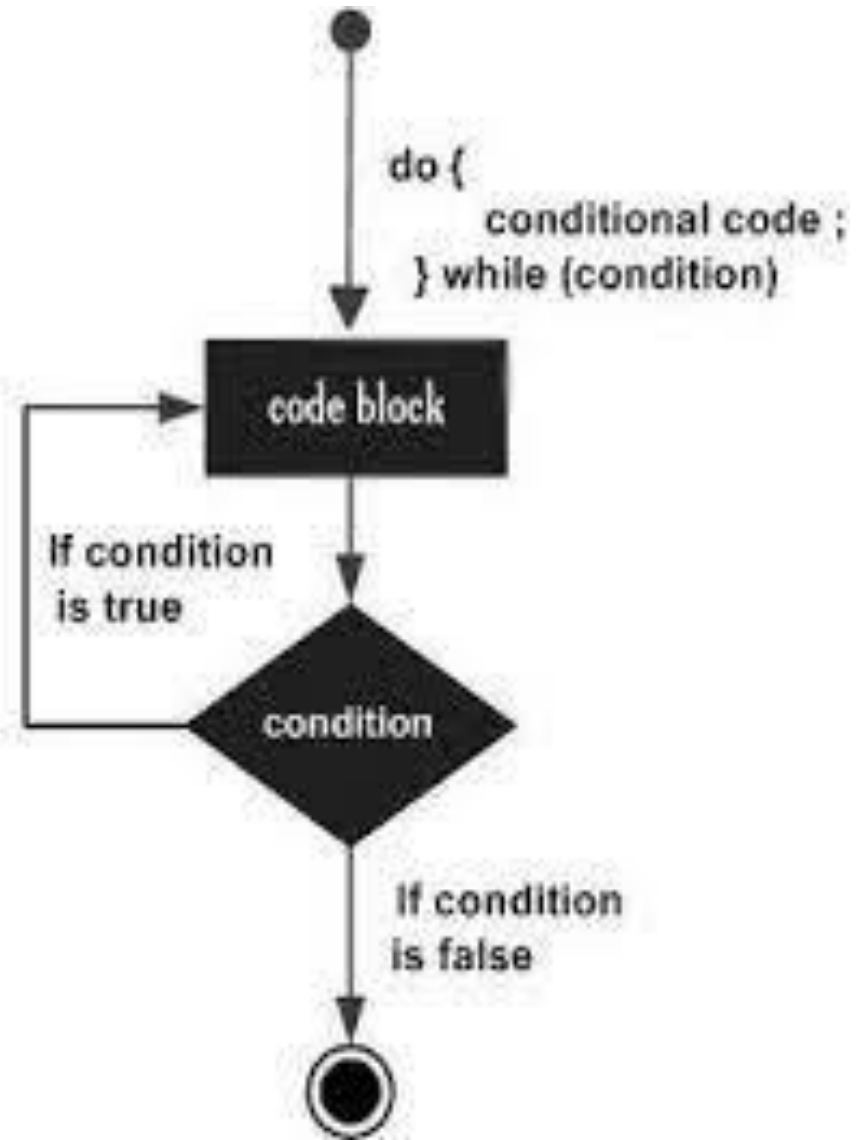
```
while (true){  
    // your code goes here  
}
```

# Câu lệnh while và do-while

- Ngôn ngữ lập trình Java cũng cung cấp một câu lệnh **do-while**, được thể hiện như sau:

```
do {  
    statement(s)  
} while (expression);
```

## DoWhileDemo



# Vòng lặp for

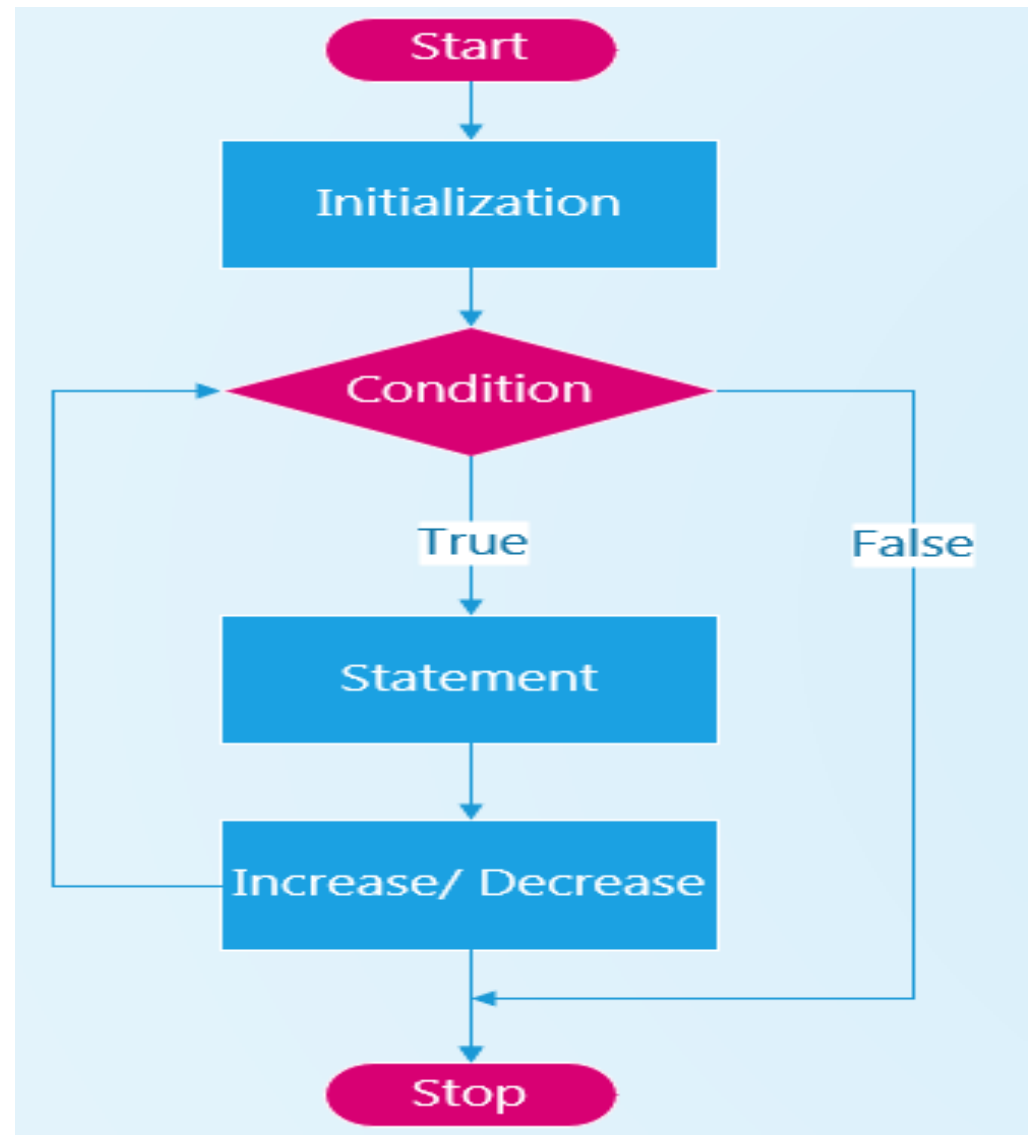
- Cú pháp  
**for**(initialization;  
termination;  
increment)

{

statement(s)

}

Condition: boolean



# Vòng lặp for

- Biểu thức khởi tạo **initialization** khởi đầu vòng lặp; nó thực thi một lần, giống như bắt đầu vòng lặp.
- Khi biểu thức kết thúc **termination** tính ra giá trị false, vòng lặp kết thúc.
- Biểu thức thay đổi **increment** là cần thiết sau mỗi lần thực hiện lặp, nó làm cho biểu thức điều kiện tăng hoặc giảm giá trị.
- Ví dụ **ForDemo**

# Vòng lặp for

- Ba biểu thức điều kiện trong vòng lặp for là tùy chọn; một vòng lặp vô hạn được tạo ra như sau:

```
// infinite loop
for ( ; ; ) {
    // your code goes here
}
```

- Câu lệnh for và các hình thức lặp khác thường được dùng trong kiểu **Collections** và kiểu **mảng (arrays)**
- Ví dụ: **EnhancedForDemo**

# Câu lệnh phân nhánh chương trình.

## ➤ Câu lệnh **break**.

- ❖ Câu lệnh **break** có hai hình thức: gán nhãn và không gán nhãn.
- ❖ Bạn thấy hình thức không án nhãn trong phần đã bàn ở trước của câu lệnh **switch**. Bạn có thể sử dụng **break** không gán nhãn để kết thúc một câu lệnh lặp **for**, **while** hoặc **do-while** như trong ví dụ **BreakDemo**.

# Câu lệnh phân nhánh chương trình.

## ➤ Câu lệnh **continue**.

- ❖ Câu lệnh **continue** bỏ qua vòng lặp hiện tại của các câu lệnh lặp **for**, **while** hoặc **do-while**.
- ❖ Hình thức không gán nhãn nhảy đến cuối thân vòng lặp và tính toán giá trị biểu thức điều kiện (boolean) để điều khiển vòng lặp.
- ❖ Ví dụ **ContinueDemo** nhảy đến một biến **String**, đếm số lần xuất hiện của ký tự “p”. Nếu ký tự hiện tại không là p, câu lệnh **continue** sẽ bỏ qua phần còn lại của vòng lặp và chuyển sang ký tự tiếp theo. Nếu nó là “p” chương trình sẽ tăng đếm ký tự.



# Câu lệnh phân nhánh chương trình.

## ➤ Câu lệnh **continue**

- ❖ Một câu lệnh nhãn **continue** bỏ qua vòng lặp hiện tại và thoát ra vị trí đánh dấu nhãn. Chương trình **ContinueWithLabelDemo** sử dụng vòng lặp lồng để tìm chuỗi con trong một chuỗi khác.
- ❖ Hai vòng lặp yêu cầu: một vòng lặp chạy qua chuỗi con và một vòng lặp chạy qua xâu tìm kiếm.



# Tổng kết về các câu lệnh luồng điều khiển

- Câu lệnh điều khiển **if-then** khá cơ bản trong tất cả các câu lệnh luồng điều khiển.
- Nó nói với chương trình thực thi một phần code chỉ khi kiểm biểu thức điều kiện **if** có giá trị **true**.
- Câu lệnh if-then-else cung cấp nhánh thức hai được thực thi khi mệnh đề if là false. Khác với if-then và **if-then-else**, câu lệnh **switch** cho phép một số đường được thực thi.



# Tổng kết về các câu lệnh luồng điều khiển

- Câu lệnh **while** và **do-while** tiếp tục thực thi khối lệnh while trong khi điều kiện là true. Sự khác nhau giữa **do-while** và **while** đó là **do-while** tính toán biểu thức điều kiện ở cuối vòng lặp thay vì đầu vòng lặp.
- Vì vậy, câu lệnh sẽ điều khiển thực hiện câu lệnh nằm trong khối do ít nhất một lần. Câu lệnh for cung cấp một cách gọn nhẹ hơn lặp trong phạm vi giá trị.

# Bài tập thực hành.

➤ Bài 1. Trong chương trình sau, hãy giải thích vì sao số 6 được in ra 2 lần.

```
class PrePostDemo {  
    public static void main(String[] args){  
        int i = 3;  
        i++;  
        System.out.println(i);    // "4"  
        ++i;  
        System.out.println(i);    // "5"  
        System.out.println(++i);  // "6"  
        System.out.println(i++);  // "6"  
        System.out.println(i);    // "7"  
    }  
}
```

# Bài tập thực hành.

## ➤ Xét đoạn code

```
if (aNumber >= 0)
    if (aNumber == 0)
        System.out.println("first string");
    else System.out.println("second string");
System.out.println("third string");
```

- Đầu ra của chương trình là gì nếu aNumber là 3.
- Viết một chương trình kiểm tra nội dung chương trình với aNumber là 3. Đầu ra khi chạy chương trình là gì? Dự đoán của bạn? Giải thích tại sao đầu ra lại như vậy. Trong trường hợp khác. Mô tả luồng điều khiển trong chương trình.
- Sử dụng dấu cách và ngắt đoạn để cho đoạn mã nguồn đó dễ đọc và dễ hiểu hơn.
- Sử dụng dấu ngoặc nhọn để code tường minh hơn.

# Bài tập thực hành.

- Bài tập 3. Dùng 3 cách: Câu lệnh if, câu lệnh if else, switch case:
- **Đọc** số trong phạm vi từ 0 đến 9. (lấy phần dư của ngày sinh với 10.): 3->ba
- Bài tập 4. Tính giai thừa của tháng sinh: Sử dụng 3 cách, do, do-while, for. Mô tả từng bước thực hiện, và kết quả.
- Bài tập 5. Vẽ hình tam giác vuông bằng dấu sao (\*).

# Bài tập thực hành.

- Bài tập 6. Vẽ hình chữ nhật bằng dấu sao(\*) có chiều cao là ngày sinh, và chiều rộng là tháng sinh của bạn.
- Bài tập 7. Nhập vào một mảng số nguyên, hãy tìm số chia hết cho ngày sinh của bạn.
- Bài tập 8. Sử dụng kết quả của bài tập 3, nhập vào số bất kỳ, viết ra cách đọc số đó.
- 145 – một bốn năm/ một trăm bốn mươi năm

