Tham khảo nhanh về Java

Nguồn: Internet | Dịch: KhoaNV

Chú thích (Comment)

// Mọi nội dung trong dòng này sẽ bị bỏ qua. Đây là loại chú thích được sử dụng phổ biến.
/* Mọi thứ (có thể gồm nhiều dòng) trong khối này sẽ bị bỏ qua */.
Không phổ biến. Sử dụng để chú thích bên ngoài mã nguồn.
/** Được sử dụng để giúp tự động tạo ra javadoc dưới dạng HTML. */

Định danh\Đặt tên

- Các định danh phải bắt đầu với ký tự bảng chữ cái Alphabetic (a-z hoặc A-Z), và các ký tự tiếp theo là chữ cái, chữ số (0-9), hoặc dấu gạch dưới (_). Không sử dụng ký tự '$'.
- Các từ tiếp theo trong định danh (nếu có) nên bắt đầu với ký tự chữ cái in hoa.
- Không sử dụng các từ khóa của Java.
- Tên lớp và giao tiếp (interface) nên bắt đầu với ký tự chữ cái in hoa

  (Graphics, String, Car, Motorbike, ...).
- Tên biến và phương thức nên bắt đầu với ký tự chữ cái in thường

  (repaint(), x, ...).
- Các hằng giá trị nên sử dụng toàn bộ chữ in hoa và dùng dấu gạch dưới (_) để phân cách các từ

  (BoxLayout.X_AXIS, Math.PI, ...).

Biến cục bộ, Biến đối tượng, Biến tĩnh

Biến có thể là biến cục bộ (*local),* biến đối tượng *(instance),*hoặc biến tĩnh (*static* hay còn gọi là biến lớp). Các tham số được coi là những biến cục bộ, chúng được gán giá trị khi phương thức được triệu gọi.

	Biến cục bộ	Biến đối tượng	Biến tĩnh
Nơi khai báo?	Trong phương thức.	Trong lớp, nhưng không trong phương thức.	Trong lớp và sử dụng thêm từ khóa *static*.
Giá trị khởi tạo	Phải được gán giá trị trị trước khi sử dụng và trình biên dịch sẽ báo lỗi nếu không làm việc này.	Kiểu số: 0 Đối tượng: null Boolean: false *Hoặc được khởi tạo giá trị trong hàm tạo*.	Kiểu số: 0 Đối tượng: null Boolean: false *Hoặc được khởi tạo trong khối khởi tạo tĩnh*.
Phạm vi sử dụng	Chỉ trong phương thức chứa nó. Không có khả năng hiển thị có thể được khai báo.	private: Chỉ các phương thức của lớp. *(Mặc định):* Tất cả các phương thức thuộc các lớp cùng gói (package). public: Bất cứ đâu đều có thể truy xuất. protected: Lớp chứa nó và tất cả các lớp con (subclass).	Giống với Biến đối tượng.
Được tạo ra khi nào?	Khi phương thức được sử dụng.	Khi một đối tượng của lớp được tạo mới.	Khi chương trình được nạp.
Được lưu trữ ở đâu?	Stack	Heap	Vùng nhớ "Bền vững"
Khi nào được giải phóng?	Khi phương thức trả về kết quả.	Khi không còn tham chiếu nào dành cho đối tượng.	Khi chương trình kết thúc.

Các kiểu dữ liệu nguyên thủy

boolean (với chỉ hai giá trị true/false)
Các kiểu số học: byte, short, char, int, long, float, double

Biểu thức

Cặp ngoặc tròn () có 3 tác dụng:

1. Nhóm để điều khiển thứ tự tính toán, hoặc để biểu thức rõ ràng. VD: (a + b) * (c - d)
2. Đặt sau tên phương thức để chứa các tham số. VD: x = sum(a, b);
3. Chứa tên kiểu dữ liệu khi tiến hành *ép kiểu.* VD: i = (int)x;

Độ ưu tiên của các toán tử

1. Toán tử có độ ưu tiên cao hơn sẽ được thực hiện trước. 2. Thứ tự sẽ từ trái qua phải nếu các toán tử có cùng độ ưu tiên, ngoại trừ: một ngôi, gán, điều kiện.	Chỉ cần nhớ thứ tự sau 1. các toán tử một ngôi *2.* * / % *3.* + - *4. các toán tử so sánh* *5.* && ‖ *6. = các toán tử gán* Sử dụng cặp ( ) cho tất cả những toán tử

Các toán tử Số học

Kết quả của các toán tử Số học là double nếu có một toán hạng là double, trái lại là float nếu có một toán hạng là float, trái lại là long nếu có một toán hạng là long, trái lại là int.

i++ Cộng thêm 1 cho i
i-- Bớt 1 từ i
n + m Phép cộng. VD 7 + 5 bằng 12, 3 + 0.14 bằng 3.14
n - m Phép trừ
n * m Phép nhân. VD 3 * 6 bằng 18
n / m Phép chia. VD 3.0 / 2 bằng 1.5, 3 / 2 bằng 1
n % m Phép chia lấy phần dư (Mod). VD 7 % 3 bằng 1

So sánh các giá trị nguyên thủy

Kết quả của các toán tử so sánh là boolean (true hoặc false).
= =, !=, <, <=, >, >=

Các toán tử Lô-gíc

Các toán hạng phải có kiểu là boolean. Kết quả là boolean.
b && c "Và". Kết quả là true nếu cả hai toán hạng đều là true, trái lại sẽ là false.
Đánh giá đoản mạch. VD (false && anything) cho kết quả false.
b ‖ c "Hoặc". Kết quả là true nếu bất cứ toán hạng nào true, trái lại sẽ là false.
Đánh giá đoản mạch. VD(true ‖ anything) cho kết quả là true.
!b "Phủ định". Kết quả là true nếu b là false, là false nếu b là true.

Các toán tử Gán

= Vế trái bắt buộc phải là một định danh\biến.
+= -= *= ...
Tất cả các toán tử hai ngôi (ngoại trừ && và ‖) đều có thể kết hợp với toán tử gán.
VD: a += 1 tương tự như a = a + 1

Ép kiểu

Ép kiểu được sử dụng khi "thu hẹp" dải giá trị nào đó. Phạm vi từ hẹp nhất đến rộng nhất của kiểu dữ liệu nguyên thủy là: byte, short, char, int, long, float, double. Các đối tượng có thể được gán mà không cần ép kiểu lên cấp cao hơn trong cây phân cấp kế thừa. Ép kiểu chỉ cần khi ép xuống cấp thấp hơn trong cây phân cấp (ép xuống).
(t)x Ép kiểu của x sang kiểu t

Toán tử Đối tượng

co.f Thành phần. Thuộc tính hoặc phương thức f của đối tượng hoặc lớp co.
x instanceof co Cho kết quả *true* nếu đối tượng tham chiếu bởi x là thể hiện của lớp co.
s + t Toán tử cộng chuỗi nếu một hoặc cả hai toán tử có kiểu là String.
x == y Cho kết quả là *true* nếu cả x và y cùng tham chiếu tới một đối tượng, trái lại sẽ là false (thậm chí cả khi giá trị của các đối tượng này giống nhau!).
x != y Phủ định của toán tử trên.
Lưu ý: So sánh các đối tượng sử dụng phương thức .equals() hoặc .compareTo()
x = y Sao chép tham chiếu đối tượng chứ không phải sao chép *đối tượng.*

Các cấu trúc điều khiển

Lệnh if

```
// lệnh if với mệnh đề true
    if(biểu_thức){
        Các lệnh  // thực hiện nếu biểu thức cho kết quả true
    }

// lệnh if với mệnh đề true và false
    if(biểu_thức){
        Các lệnh  // thực hiện nếu biểu thức cho kết quả true
    }else{
        Các lệnh  // thực hiện nếu biểu thức cho kết quả false
     }

// Lệnh if với nhiều điều kiện kiểm tra song song
    if(biểu_thức_1){
        Các lệnh  // thực hiện nếu biểu thức 1 là true
    }else if(biểu_thức_2){
        Các lệnh  // thực hiện nếu biểu thức 2 là true
    }else if(biểu_thức_3){
        Các lệnh  // thực hiện nếu biểu thức 3 là true
    }else{
        Các lệnh  // thực hiện khi các biểu thức trên là false
    }
```

Lệnh switch

switch cho phép chọn một trường hợp nào đó phụ thuộc vào giá trị nhận được (thường là số nguyên) từ biến hoặc biểu thức.

```
switch(biểu_thức){
  case c1:
      Các lệnh  // thực hiện khi biểu thức = c1
      break;
  case c2:
       Các lệnh  // thực hiện khi biểu thức = c2
       break;
  case c3:
  case c4:
  case c5: // các trường hợp cùng chung xử lý.
        // thực hiện khi biểu thức = c3, c4 hoặc c5
       Các lệnh
       break;
  default:
       Các lệnh
       // thực hiện khi mà biểu thức không bằng giá trị nào ở trên
}
```

Lệnh lặp while

```
while(biểu_thức){
    // thực hiện lặp lại các mệnh lệnh cho tới khi
    // biểu thức cho kết quả false
}
```

Lệnh lặp for

```
for(Khởi tạo biến đếm; điều kiện lặp; thay đổi biến đếm) {
    // thực hiện lặp lại các mệnh lệnh cho tới khi
    // biểu thức điều kiện lặp cho kết quả false
}
```

while và for có thể thay thế cho nhau:

```
int i = 0;
while(i<5){
   System.out.print("Hi!");
   i++;
}
```

```
for (i=0; i<5; i++) {
    System.out.print("Hi!");
}
```

Những điều khiển lặp khác

Tất cả các lệnh lặp đều có thể được đặt nhãn, vì vậy có thể dùng lệnh break và continue cho bất cứ cấp độ lồng nhau nào của vòng lặp.

```
break; //thoát khỏi vòng lặp hoặc lệnh switch gần nhất chứa nó
break label; // thoát khỏi vòng lặp được đặt nhãn (label)
continue; //tiến hành lượt lặp tiếp theo
continue label; //tiến hành lượt lặp tiếp theo với vòng lặp được đặt nhãn (label).
```

Nhãn được đặt trước vòng lặp cùng dấu hai chấm, ví dụ:

```
outer:
for (. . .) {
      . . .
      continue outer;
}
```

Ngoại lệ (Exception)

Sử dụng try...catch cho các ngoại lệ

```
try{
      . . . // các lệnh có thể dẫn tới ngoại lệ
}catch(Kiểu_ngoại_lệ x){
      . . . // các lệnh để xử lý khi có ngoại lệ
}
```

throw

```
throw đối_tượng_ngoại_lệ;
```

Sử dụng nhiều mệnh đề catch và mệnh đề finally

Thực hiện mệnh đề catch đầu tiên tương ứng với ngoại lệ được chỉ ra bởi lớp ngoại lệ hoặc các lớp ngoại lệ cha. Mệnh đề finally luôn được thực hiện (bất kể là có hay không có ngoại lệ xảy ra) vì vậy các tài nguyên sử dụng trước đó có thể được giải phóng (ví dụ, đóng tệp tin):

```
try{
      . . . // các lệnh có khả năng xảy ra ngoại lệ
}catch(kiểu_ngoại_lệ x){
      . . . // cách lệnh xử lý ngoại lệ
} catch (kiểu_ngoại_lệ x) {
      . . . // cách lệnh xử lý ngoại lệ
}finally{
      // các lệnh luôn luôn được thực hiện
      // bất kể có xảy ra ngoại lệ hay không
      . . .
}
```

Chuỗi

Cộng chuỗi

Toán tử + cho phép nối hai chuỗi lại với nhau. Nếu một toán hạng là String, toán hạng còn lại sẽ được chuyển đổi sang kiểu String và sau đó được cộng với nhau. Đây thường là cách dùng để chuyển đổi các số sang String.

Nếu một đối tượng không phải là String khi được cộng với String phương thức toString() của nó sẽ được triệu gọi. Một tiện ích dành cho việc ra soát bug đó là viết trong các lớp của bạn phương thức toString().

"abc" + "def"	"abcdef"
"abc" + 4	"abc4"
"1" + 2	"12"
"xyz" + (2+2 == 4)	"xyztrue"
1 + "2.5"	"12.5"

`i = s.length()` cho phép để xác định độ dài của chuỗi s.

So sánh chuỗi (sử dụng thay thế cho toán tử == và !=)

`i = s.compareTo(t)` được sử dụng để so sánh chuỗi s với t.
trả về <0 nếu s < t, 0 nếu s == t, >0 nếu s > t

`i = s.compareToIgnoreCase(t)` tương tự như trên nhưng không phân biệt ký tự in hoa, in thường.

`b = s.equals(t)` cho kết quả true nếu hai chuỗi có cùng giá trị.

`b = s.equalsIgnoreCase(t)` tương tự như trên nhưng không phân biệt ký tự in hoa, in thường

`b = s.startsWith(t)` cho kết quả **true** nếu chuỗi s chứa chuỗi t ở đầu

`b = s.endsWith(t)` cho kết quả **true** nếu chuỗi s chứa chuỗi t ở cuối

Tìm kiếm trong chuỗi (tất cả các phương thức "indexOf" đều trả về -1 nếu không tìm thấy kết quả)

`i = s.indexOf(t)` trả về vị trí đầu tiên xuất hiện chuỗi t trong chuỗi s.

`i = s.indexOf(t, i)` trả về vị đầu tiên trí tính từ i xuất hiện chuỗi t trong chuỗi s.

`i = s.lastIndexOf(t)` trả về vị trí cuối cùng xuất hiện chuỗi t trong chuỗi s.

`i = s.lastIndexOf(t, i)` trả về vị trí cuối cùng tính từ i xuất hiện chuỗi t trong chuỗi s.

Lấy một phần của chuỗi

`c = s.charAt(i)` trả về ký tự tại vị trí i trong chuỗi s.

`sl= s.substring(i)` trả về một chuỗi con của chuỗi s tính từ vị trí i đến cuối.

`sl= s.substring(i, j)` trả về một chuỗi con của s tính từ vị trí i đến ngay trước vị trí j.

Tạo mới một chuỗi từ chuỗi ban đầu

`sl= s.toLowerCase()` tạo một chuỗi mới ở dạng in hoa toàn bộ

`sl= s.toUpperCase()` tạo một chuỗi mới ở dạng in thường toàn bộ

`sl= s.trim()` cắt bỏ các ký tự trắng ở đầu và cuối chuỗi

`sl= s.replace(cs2, cs3)` thay thế tất cả các chuỗi con cs2 trong s bằng chuỗi cs3

StringBuilder

Khả năng sửa đổi nhanh hơn so với String, sử dụng bộ nhớ và CPU hiệu quả hơn.

`sb = new StringBuilder()` Tạo một StringBuilder rỗng

`sb = new StringBuilder(s)` Tạo một StringBuilder từ chuỗi s.

`sb = sb.append(x)` Thêm x (bất kể kiểu dữ liệu)vào cuối sb.

`sb = sb.insert(offset, x)` Chèn x (bất kể kiểu dữ liệu) vào sb tại vị trí offset.

`sb = sb.setCharAt(index, c)` Thay thế ký tự tại vị trí index bằng ký tự c.

`sb = sb.deleteCharAt(i)` Xóa ký tự tại vị trí i.

`sb = sb.delete(beg, end)` Xóa các ký tự từ vị trí beg tới vị trí end.

`sb = sb.reverse()` Đảo ngược nội dung.

`sb = sb.replace(beg, end, s)` Thay thế các ký tự từ vị trí beg tới vị trí end bằng s.

Có các phương thức indexOf, lastIndexOf, charAt, equals, substring giống như String!

Mảng

Sử dụng để thao tác với nhiều phần tử dữ liệu, có thể là dữ liệu nguyên thủy hoặc các đối tượng. Tất cả các phần tử phải cùng kiểu dữ liệu.Mảng không có khả năng mở rộng!

Ví dụ:

```
int [] scores; // Khai báo scores là mảng các số nguyên.
scores = new int[12]; // Khởi tạo vùng nhớ cho mảng với 12 phần tử.
int[] scores = new int[12]; // Kết hợp cả khai báo và khởi tạo.
```

Khởi tạo một mảng

Nếu các phần tử của mảng không được gán giá trị khởi tạo, chúng sẽ được khởi tạo là 0 với các mảng số, null với các mảng tham chiếu đối tượng, và false với các mảng boolean.

Tạo mảng vào khởi tạo giá trị cho phần tử mảng với một dòng lệnh:

```
String[] names = {"Mickey", "Minnie", "Donald"};
```

Hoặc có thể tách thành nhiều dòng lệnh:

```
String[] names = new String[3];
names[0] = "Mickey";
names[1] = "Minnie";
names[2] = "Donald";
```

Truy cập phần tử mảng

```
scores[5] = 86; // Gán cho phần tử có chỉ số là 5 giá trị 86.
scores[i]++; // Tăng một đơn vị cho phần tử có chỉ số là i.
```

Duyệt lần lượt qua các phần tử mảng

Kích thước của một mảng có thể được xác định bằng cách sử dụng thuộc tính *length* của chúng, VD, `scores.length`

```
// Sử dụng vòng lặp for chuẩn.
int[] scores = new int[12];
...Khởi tạo các giá trị cho mảng scores
int total = 0;
for (int i = 0; i <scores.length; i++)
{
   total += scores[i];
}
```

```
// Sử dụng vòng lặp for nâng cao.
int[] scores = new int[12];
...Khởi tạo các giá trị cho mảng scores
int total = 0;
for(int scr : scores){
    total += scr;
}
```

Mảng hai chiều

Hầu hết được xử lý với vòng lặp for lồng nhau. Ví dụ:

```
static final int ROWS = 2; static
final int COLS = 4;
…
int[][] a2 = new int[ROWS][COLS];
// ...   Hiển thị các phần tử mảng trong khung hình chữ nhật
for(int i =0; i < ROWS; i++){
     for(int j = 0; j < COLS; j++){
          Systern.out.print("  " + a2[i][j]);
     }
     Systern.out.println("");
}
```

Scanner

Tác dụng chính của java.util.Scanner là giúp đọc các giá trị từ System.in hoặc từ tệp tin.

`sc = new Scanner(System.in);` Sử dụng để đọc dữ liệu từ System.in

`sc = new Scanner(s);` Sử dụng để đọc giá trị từ chuỗi s.

Các phương thức lấy dữ liệu phổ biến nhất

`s = sc.next()` Trả về "token" tiếp theo, nhiều hơn hoặc ít là một "word".

`s = sc.nextLine()` Trả về toàn bộ một dòng dữ liệu dưới dạng một chuỗi.

`x = sc.nextXYZ()` Trả về giá trị có kiểu là XYZ:
Int, Double, Boolean, Byte, Float, Short

`b = sc.hasNext()` Trả về true nếu còn "token" để đọc.

`b = sc.hasNextLine()` Trả về true nếu còn dòng dữ liệu để đọc.

`b = sc.hasNextXYZ()` Trả về true nếu loại còn dữ liệu XYZ để đọc.

Nhập\Xuất tệp tin Văn bản

Ví dụ:

```
public static void copyFile(File fromFile,
                            File toFile) throws IOException {
  Scanner freader = new Scanner(fromFile);
  BufferedWriter writer = new BufferedWriter(new FileWriter(toFile));
  //... Lặp cho tới khi vẫn còn dòng dữ liệu từ tệp đầu vào.
  String line = null;
  while (freader.hasNextLine()) {
      line = freader.nextLine();
      writer.write(line);
      writer.newLine(); // Viết sang dòng khác trong tệp tin.
  }
  //... Đóng bộ đọc và ghi tệp tin.
  freader.close(); // Đóng để bỏ khóa đối với tệp tin.
  writer.close(); // Đóng để bỏ khóa đối với tệp tin và đẩy toàn bộ dữ liệu xuống ổ đĩa.
}
```