**Khái niệm và quy ước**

* Toán tử: hiểu đơn giản là phép toán.
* Toán tử 1 ngôi: toán tử chỉ cần 1 toán hạng là có thể hoàn tất biểu thức.
* Toán tử 2 ngôi: toán tử cần có 2 toán hạng trái và phải phép toán thì mới hoàn thành được biểu thức.
* Nếu phép toán có nhiều hơn 1 kí tự thì phải viết liền các kí tự sát nhau.
* Luôn phân tách giữa toán tử hai ngôi và các toán hạng của nó bằng 1 dấu cách cho chương trình dễ đọc.

**Các toán tử số học: +, -, \*, /, %, ++, —**

**Toán tử +, –**

* Thứ tự ưu tiên thực hiện từ trái qua phải
* Nếu là chúng được sử dụng như toán tử 1 ngôi thì phải đặt trước toán hạng.
* Kiểu của toán hạng phải là các kiểu số hoặc kiểu lớp bao của kiểu số nếu không chương trình sẽ báo lỗi.
* Ví dụ:

int a = 5;

int b = -a; *// dùng - như toán tử 1 ngôi*

int c = +b; *// dùng + như toán tử 1 ngôi*

* Nếu sử dụng như toán tử hai ngôi thì hai toán hạng trái phải của toán tử phải là kiểu số hoặc kiểu có thể chuyển sang số
* Khi thực hiện cộng trừ có thể xảy ra vấn đề tràn số nếu giá trị kết quả lớn hơn hoặc nhỏ hơn khả năng lưu trữ của kiểu.
* Nếu toán hạng có chứa kiểu float hoặc double thì:
  + Nếu 1 trong hai toán hạng có giá trị là NaN thì kết quả là NaN. NaN là một hằng số trong lớp bao của kiểu float, double ám chỉ giá trị đáng lẽ ra là số nhưng lại không phải số. Cú pháp lấy NaN: Float.NaN hoặc Double.NaN.
  + Tổng của hai giá trị vô cực trái dấu là NaN
  + Tổng của hai giá trị vô cực cùng dấu là giá trị vô cực cùng dấu
  + Tổng của một giá trị giới hạn và một giá trị vô cực là vô cực
  + Tổng của hai giá trị 0 trái dấu là +0
  + Tổng của giá trị 0 và giá trị khác 0 là giá trị khác 0
  + Tổng của hai giá trị cùng độ lớn nhưng trái dấu là +0
* Mặc dù có thể xảy ra tràn số, mất thông tin nhưng việc thực hiện các phép toán trên không gây ra việc ném các ngoại lệ run-time exception

**Trường hợp phép + với String:**

* Nếu áp dụng phép + với ít nhất 1 toán hạng là kiểu String thì kết quả là một String. Do đó toán tử + còn gọi là toán tử nối chuỗi kí tự
* Ví dụ: String str = “Hello”; str + 5 + 6 sẽ cho kết quả “Hello56” không phải “Hello11”
* Để có kết quả chính xác ta có thể gộp nhóm với cặp () ví dụ str + (5 + 6)
* Cụ thể về lớp String sẽ tìm hiểu trong bài học về sau

**Toán tử \*, /, %:**

* Thứ tự ưu tiên thực hiện biểu thức là từ trái qua phải
* Đây là các toán tử hai ngôi với hai toán hạng bắt buộc phải là kiểu số hoặc kiểu lớp bao của các kiểu số nếu không chương trình sẽ báo lỗi
* Kết quả thực hiện của các phép toán trên thường được lưu vào một biến có kiểu là kiểu lớn nhất trong kiểu của các toán hạng. Ví dụ giá trị kiểu int nhân với kiểu long thì kết quả sẽ ở kiểu long vì kiểu long có kích thước lớn hơn int tính theo số bit hoặc số byte.
* Nếu hai toán hạng ở kiểu nguyên và xảy ra việc tràn số khi tính toán, giá trị kết quả thu được có thể khác dấu với giá trị đúng khi thực hiện cùng phép toán đó.
* Nếu nhân mà 1 trong các toán hạng là kiểu float hoặc double thì:
  + Nếu 1 toán hạng là NaN thì kết quả là NaN.
  + Nếu kết quả không phải NaN thì kết quả cho dấu + khi hai toán hạng cùng dấu và ngược lại cho dấu âm khi hai toán hạng trái dấu.
  + Nhân giá trị NaN với vô cực cho kết quả là NaN.
  + Nhân giá trị giới hạn với giá trị vô cực cho kết quả vô cực có dấu được xét theo quy tắc nhân hai số bình thường.
  + Nếu cả hai toán hạng đều khác vô cực và NaN thì kết quả thực hiện theo quy tắc nhân bình thường và lưu vào kiểu lớn nhất trong kiểu của hai toán hạng.

**Phép chia /:**

* Nếu tử số và mẫu số của phép chia đều là giá trị kiểu nguyên, kết quả sẽ làm tròn về phần nguyên của phép chia.
* Nếu cả hai toán hạng của phép chia cùng là kiểu nguyên và mẫu số bằng 0 thì sẽ xảy ra ngoại lệ chia cho 0: ArithmeticException.
* Nếu ít nhất 1 trong 2 toán hạng là kiểu float hoặc double thì:
  + Nếu 1 trong hai toán hạng là NaN thì kết quả là NaN.
  + Nếu kết quả khác NaN thì dấu của kết quả được xác định theo quy tắc nhân hai số có dấu trong toán học.
  + Chia vô cực cho vô cực được NaN.
  + Chia vô cực cho giá trị giới hạn được giá trị vô cực và dấu xác định theo quy tắc nhân.
  + Chia giá trị giới hạn cho vô cực được giá trị 0 có dấu xác định theo quy tắc nhân.
  + Chia 0 cho 0 được NaN. Nếu tử số là 0 và mẫu số khác 0 thì dấu xác định theo quy tắc nhân.
  + Chia giá trị khác 0 cho mẫu số bằng 0 được giá trị vô cực có dấu xác định theo quy tắc nhân.
  + Các trường hợp khác tính toán bình thường, kết quả làm tròn chữ số cuối cùng đến giá trị gần nhất, tức làm tròn lên nếu giá trị sau nó >= 0.5 và ngược lại.
* Phép chia có thể gây tràn số và khi đó giá trị cũng như dấu của kết quả sẽ không như mong muốn ban đầu.
* Dù có thể xảy ra việc tràn số và mất thông tin, chia cho 0 nhưng việc thực hiện\*/ không ném ngoại lệ tại thời điểm chạy chương trình.

**Phép chia lấy dư %:**

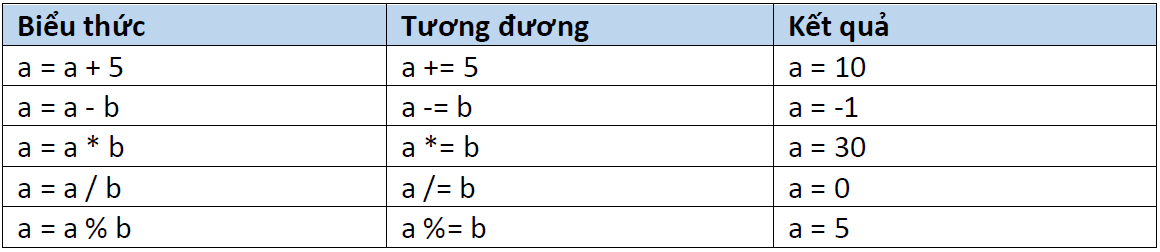
* Là toán tử hai ngôi.
* Lấy phần dư sau khi thực hiện phép chia /.
* Áp dụng cho cả kiểu số thực.
* Lưu ý mẫu số khác 0 để đảm bảo không xảy ra ngoại lệ chia cho 0.
* Dấu của kết quả chỉ phụ thuộc vào dấu của tử số.
* Nếu thao tác với số thực thì:
  + Nếu 1 trong hai toán hạng là NaN thì kết quả là NaN.
  + Nếu kết quả khác NaN dấu chỉ phụ thuộc dấu của tử số.
  + Nếu tử số là vô cực hoặc mẫu số là 0 hoặc cả hai thì kết quả là NaN.
  + Nếu tử số là giá trị giới hạn và mẫu là giá trị vô cực thì kết quả là tử số.
  + Nếu tử số bằng 0 và mẫu là giá trị khác 0 thì kết quả là tử số.
  + Các trường hợp khác lấy dư bình thường.
* Thực hiện thao tác với phép toán này cũng không xảy ra ngoại lệ trong thời gian chạy chương trình.

**Toán tử ++, –:**

* Là toán tử 1 ngôi thực hiện việc tăng hoặc giảm giá trị của một số đi 01 đơn vị.
* Có thể đặt trước hoặc sau toán hạng đều được.
* Toán hạng của hai toán tử này thường là các kiểu số hoặc kiểu lớp bao của các kiểu số nếu không chương trình sẽ báo lỗi.
* Kiểu của kết quả phép toán ++ hoặc — sẽ có kiểu trùng với kiểu của biến.
* Các phép toán này chỉ ảnh hưởng đến giá trị của biến.
* Các hằng số hay các biến có từ khóa final trong khai báo không thể áp dụng với các toán tử này.

**Các toán tử gán: =, +=, -=, \*=, /=, %=**

* Các toán tử gán là toán tử hai ngôi dùng để gán giá trị cho một biến cùng kiểu hoặc có kiểu tương thích
* Các biến kiểu lớn hơn luôn có thể chứa được giá trị của các kiểu nhỏ hơn. Thứ tự này là byte -> short -> int -> long -> float -> double.
* Nếu muốn thực hiện gán giá trị của kiểu lớn hơn cho kiểu nhỏ hơn, ta phải ép kiểu và chấp nhận mất giá trị. Nội dung này sẽ trình bày chi tiết trong bài tiếp theo.
* Thứ tự ưu tiên sẽ được thực hiện từ phải qua trái tức là nếu a = b = c thì đầu tiên gán c cho b sau đó gán b cho a.
* Toán tử gán có thể kết hợp với các toán tử số học, toán tử thao tác với bit(không học, bạn đọc có thể tự tìm hiểu thêm).
* Toán hạng trái của phép gán luôn là biến không phải hằng số, nếu là hằng số chỉ gán được duy nhất lần đầu tiên.
* Toán hạng phải của phép gán có thể là các biến, hằng số, các giá trị cụ thể của kiểu tương ứng, biểu thức, …
* Sau khi thực hiện phép gán thành công thì giá trị của biểu thức gán là giá trị của toán hạng trái. Nếu trong quá trình gán mà xảy ra lỗi thì kết quả của toán hạng trái không thay đổi
* Sau đây là bảng mô tả phép gán kết hợp với các toán tử số học với a = 5, b = 6:

[](https://braniumacademy.net/wp-content/uploads/2020/08/k%E1%BA%BFt-h%E1%BB%A3p-ph%C3%A9p-g%C3%A1n-v%E1%BB%9Bi-to%C3%A1n-t%E1%BB%AD-s%E1%BB%91-h%E1%BB%8Dc.png)

**Các toán tử so sánh: >, >=, <, <=, ==, !=**

* Là các toán tử hai ngôi dùng để so sánh mối quan hệ giữa hai toán hạng được áp dụng
* Thường chỉ sử dụng các phép so sánh với các kiểu số nên các toán hạng của các phép so sánh phải là kiểu có thể chuyển về kiểu số nếu không thì chương trình sẽ bị lỗi
* Kết quả của phép so sánh luôn ở kiểu boolean theo đúng mối quan hệ về giá trị độ lớn của hai toán hạng
* Việc so sánh hai số thực luôn cho kết quả chính xác
* Với số thực:
  + Nếu 1 trong hai toán hạng là NaN thì kết quả là false
  + Tất cả các giá trị khác theo thứ tự: âm vô cực < giá trị giới hạn < dương vô cực
  + -0.0 và 0.0 là tương đương nhưng phương thức min, max của lớp Math coi -0.0 < 0.0

**Với toán tử == và !=:**

* Thứ tự thực hiện từ trái sang phải
* Hai toán hạng phải cùng kiểu để đảm bảo không bị lỗi
* Kết quả ở kiểu boolean
* Phép == cho kết quả true khi và chỉ khi hai biểu thức trái phải của nó là tương đương
* Phép != chỉ cho kết quả true khi và chỉ khi biểu thức trái và phải của nó không tương đương nhau về mặt giá trị
* Biểu thức a != b tương đương !(a == b)
* Với số thực:
  + Nếu 1 trong các toán hạng là NaN thì kết quả so sánh == luôn false nhưng != là true. Nói cách khác, x != x true khi và chỉ khi x có giá trị là NaN
  + Có thể dùng Float.isNaN hoặc Double.isNaN để kiểm tra giá trị của biến nào đó có phải NaN không
  + 0 âm và 0 dương được coi là tương đương nhau, tức -0.0 == 0.0 cho kết quả true
  + Các trường hợp khác đánh giá như bình thường

**Các toán tử logic: &&, ||, !**

**Toán tử và(&&):**

* Thứ tự ưu tiên thực hiện biểu thức từ trái qua phải
* Là toán tử hai ngôi với hai toán hạng trái phải cùng có kiểu boolean hoặc Boolean hoặc chương trình sẽ báo lỗi
* Toán hạng phải chỉ được đánh giá khi toán hạng trái là true
* Kết quả thu được luôn ở kiểu boolean, tức chỉ có thể là true hoặc false

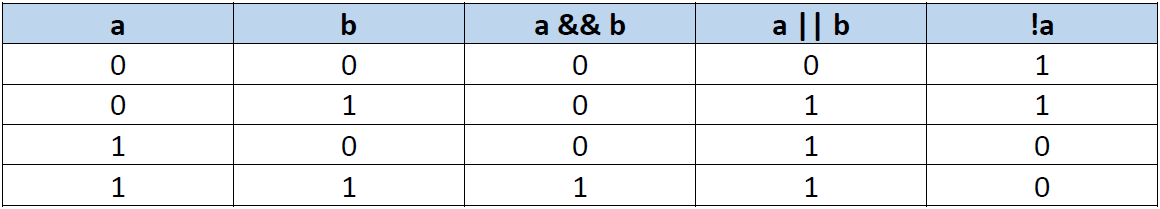
**Toán tử hoặc(||):**

* Thứ tự ưu tiên thực hiện biểu thức từ trái qua phải
* Là toán tử hai ngôi với cả hai toán hạng trái và phải cùng kiểu boolean hoặc Boolean. Ngược lại chương trình sẽ báo lỗi.
* Toán hạng phải chỉ được đánh giá khi toán hạng trái là false
* Kết quả thu được là true hoặc false

**Toán tử phủ định(!):**

* Toán tử ! luôn đứng trước một toán hạng có kiểu boolean hoặc Boolean nếu không thì chương trình sẽ báo lỗi
* Kết quả của biểu thức với toán tử ! luôn có kiểu boolean
* Với toán hạng x có kiểu boolean thì:
  + Nếu x = true, !x = false
  + Nếu x = false, !x = true

Bảng tóm tắt các giá trị chân lý với 1 tương đương true, 0 tương đương false:

[](https://braniumacademy.net/wp-content/uploads/2020/08/b%E1%BA%A3ng-gi%C3%A1-tr%E1%BB%8B-ch%C3%A2n-l%C3%BD.png)Như vậy ta đã hoàn thành bài học về các toán tử trong Java. Đây là phần có nhiều điều cần lưu ý. Khi nào cần tham khảo bạn có thể quay lại bài này.