**#1 Câu lệnh rẽ nhánh**

**if**

Các bạn nhìn qua ví dụ này:

public static void main(String[] args){

// khai bao so nguyen

int a = 10;

// Kiểm tra xem a có bằng 9 không

if(a == 9){

// nếu bằng 9, in ra màn hình "Hello"

System.out.println("Hello");

}

// Kiểm tra xem a có bằng 9 không

if(a == 10){

// nếu bằng 10, in ra màn hình "World"

System.out.println("World");

}

// Kết quả trên màn hình:

// World

}

Nếu các bạn chạy ví dụ ở trên thì sẽ thấy kết quả trên màn hình chỉ có chữ World. Lí do thì bạn biết rồi phải không 😗 Cái mình muốn các bạn để ý ở đây là cái dấu ==. Sao lại == 🤔 mà không phải =

Thì các cần biết như sau, câu lệnh if là một câu lệnh điều kiện, và nhận vào là một điều kiện true hoặc false. Có cú pháp như sau:

if ([điều kiện]){

// Thực hiện đoạn code nếu [điều kiện] là `true`. Nếu `false` bỏ qa đi xuống dưới.

}

// Tiếp tục thực hiện đoạn code phía dưới

Lúc này nếu bạn viết if(a = 5) như Sách giáo khoa toán thì sẽ nhận về kết cục cay đắng 😢 vì trong [Bài #2](https://loda.me/java-basic-2-bien-pham-vi-kieu-du-lieu-toan-tu-trong-java-loda1553050622971) mình có nói, dấu = là phép gán. Tức là gán cho a giá trị bằng 5 chứ không phải ý nghĩa là so sánh a với 5 để mà trả về là true hay false nữa.

Vậy đấy, nên để so sánh bạn cần dùng toán tử quan hệ mình liệt kê dưới đây:

* ==: Kiểm tra 2 toán hạng có bằng nhau không? (if(a==b))
* !=: Kiểm tra 2 toán hạng có khác nhau không? (if(a!=b))
* >: Kiểm tra toàn hạng A có lớn hơn B không? (if(a>b))
* <: Kiểm tra toàn hạng A có nhỏ hơn B không? (if(a<b))
* >=: Kiểm tra toàn hạng A có lớn hơn hoặc bằng B không? (if(a>=b))
* <=: Kiểm tra toàn hạng A có nhỏ hơn hoặc bằng B không? (if(a<=b))

Tất cả toán tử quan hệ ở trên, khi thực hiện xong nó sẽ trả về là kiểu boolean, nên bạn có thể gán nó vào một biến bất kỳ, như lày:

int a = 5;

int b = 6;

boolean result = a == b; // false

System.out.println("Result: " + result);

// Kết quả in ra trên màn hình:

// "Result: false"

if(result){ // viết tắt của if(result == true)

System.out.println("Result is true");

}

Đến đây, có thể nói câu lệnh if thực chất nhận vào một giá trị boolean.

**else**

Tiếp theo, chúng ta tới với dạng đầy đủ của if chính là cấu trúc if else.

if ([điều kiện]){

// Thực hiện đoạn code nếu [điều kiện] là `true`.

}else{

// Thực hiện đoạn code trong này nếu [điều kiện] là `false`

}

//Các đoạn code ở dưới thực hiện bình thường sau khi if hoặc else diễn ra

Ví dụ:

int a = 5;

if ( (a + 2) == 7 ){

System.out.println("Bằng 7");

// Sử dụng biến `a` ở ngay trong scope {} của `if`,, như bài #2 mình có nói, biến được sử dụng trong các scope con hoặc bằng cấp

System.out.println("Giá trị lúc này của a = " + a);

}else{

System.out.println("Khác 7");

System.out.println("Giá trị lúc này của a = " + a);

int b = 7; // Tạo ra 1 biến b trong else

}

b = 50; // Lỗi, không biết b là gì, vì b ở scope nhỏ hơn, bên ngoài không hiểu.

Ví dụ trên mình vừa cho các bạn thấy cách sử dụng if else vừa củng cố lại kiến thức của [Bài #2](https://loda.me/java-basic-2-bien-pham-vi-kieu-du-lieu-toan-tu-trong-java-loda1553050622971) luôn.

**Toán tử logic**

Toán tử logic là những toán tử giúp chúng ta kết hợp nhiều [điều kiện] lại với nhau.

Ví dụ mình nói: "Nếu ab = 3 VÀ ac = 4 VÀ bc = 5 thì abc là tam giác vuông"

Thì trong code cần viết chương trình như thế nào?

Cách 1: Sử dụng if thông thường.

int ab = 3;

int ac = 4;

int bc = 5;

if(ab == 3){

if(ac == 4){

if(bc == 5){

System.out.println("abc là tam giác cực vuông");

}

}

}

Ohhh loooo, my eyes 😱 tôi đang nhìn cái rì thế này. Một chương trình là có vài trăm cái điều kiện chắc chết.

Cách 2: Sử dụng if và toán tử logic

int ab = 3;

int ac = 4;

int bc = 5;

// Nếu ab = 3 VÀ ac = 4 VÀ bc = 5

if(ab == 3 && ac == 4 && bc==5){

// thì abc là tam giác vuông

System.out.println("abc là tam giác cực vuông");

}

Rất gì và này nọ phải không :))) đọc mồm sao, viết code y xì vậy, quá hay, quá tuyệt vời. 😱😜

Các bạn nhìn ví dụ cũng đoán ra && chính là toán tử logic đại diện cho khái niệm AND. Chúng ta có tất cả các loại toán tử logic như sau:

* &&: AND
* ||: OR
* !: NOT

Mục tiêu của các toán tử logic là tác động lên các biểu thức boolean để cho ra một biến boolean mới.

**Phép AND (&&)**

Phép && hoạt động theo nguyên tắc, chỉ cần có 1 cái sai, thì tất cả đều sai hay Tất cả đều phải đúng, mới là đúng

Nếu "A đúng và B đúng và C sai thì kết quả vẫn là sai"

// Bạn chạy thử xem nó đi vào phần nào nhé

if(true && true && true && false){

System.out.println("true");

}else{

System.out.println("false");

}

**Phép OR (||)**

Phép || thì rất dễ dãi, Chỉ 1 cái đúng là đụ

// Bạn chạy thử xem nó đi vào phần nào nhé

if(false || false || true || false){

System.out.println("true");

}else{

System.out.println("false");

}

**Phép NOT (!)**

Phép ! làm phủ định giá trị của biểu thức, nếu nó đang true thì biến nó thành false và ngược lại.

int a = 7;

if(!(a == 7)){ // (a==7) => true gặp thằng ! lại bị chuyển thành false. => vào vế else

System.out.println("Đáng nhẽ ra nên vào đây");

}else{

System.out.println("But nope, nó lại vào đây");

}

**#2 Hàm (Function)**

**Giới thiệu**

Cùng nhìn vào ví dụ này nhé các bạn:

int a = 5 + 6;

System.out.println("In a ra màn hình: " + a);

a = 1 + 6;

System.out.println("In a ra màn hình: " + a);

a = 2 + 6;

System.out.println("In a ra màn hình: " + a);

Đây là một chương trình tính tổng 2 số rồi in ra màn hình. Tất nhiên hiện tại chúng ta chưa học cách nhập xuất trong Java nên sẽ fix cứng trong code để làm ví dụ.

Các bạn thấy là chúng ta phải viết việc a = xx + yy 3 lần và viết System.out.println("In a ra màn hình: " + a); 3 lần. Việc này thực sự đã lặp lại các bất cập trong việc viết code khi phải thực hiện một chức năng lặp đi lặp lại nhiều lần. Để giảm thiểu việc viết lại code nhiều lần chúng ta có khái niệm `Hàm (Function);

Các bạn nhìn ví dụ, mình sẽ giải thích:

public class Calculation {

public static void main(String[] args){

f(5,6);

f(2,3);

f(1,10);

}

public static void f(int x, int y){

int a = x + y;

System.out.println("In a ra màn hình: " + a);

}

}

// Kết quả khi chạy:

// In a ra màn hình: 11

// In a ra màn hình: 5

// In a ra màn hình: 11

Từ ví dụ, chúng ta sẽ liên tưởng tới khái niệm toàn học hồi cáp 2, hàm f(x). Là cách các nhà toán học ký hiệu cho một hàm số nào đó, sau chỉ cần thay x vào là được, còn về cơ bản nó vẫn là một hàm số và thay đổi tham số đầu vào. Nguyên tắc sử dụng hàm để tiết kiệm thời gian viết cũng như đóng gói chức năng là như vậy.

**Cách khai báo**

Cách khai báo một phương thức như sau:

[kiểu\_truy\_cập] [kiểu\_trả\_về] [tên\_phương\_thức] ([danh\_sách\_tham\_số]){}

ví dụ:

public static void f(int x, int y){

//Code của bạn

}

public static void main(String[] args){

}

Và khai báo ở ngoài hàm main(). Tới đây, bạn hiểu main() cũng là một hàm (function). Tuy nhiên nó đặc biệt vì cú pháp của nó là cố định và được Java tìm tới để đọc đầu tiên.

1 - [kiểu\_truy\_cập]:

Trong ví dụ trên [kiểu\_truy\_cập] chính là vế public static. Nó định nghĩa phạm vi hàm được sử dụng. chúng ta sẽ tìm hiểu ở các bài sau nhé các bạn, bây giờ bạn hãy mặc định sử dụng public static ở trước mỗi hàm khai báo để có thể sử dụng được nhé. Ở bài này, chúng ta tạm hiểu với nhau: public static là "truy cập ở bất cứ đâu" tức có thể gọi hàm này ở bất kì chỗ nào.

2 - [kiểu\_trả\_về]:

Tương đương với phần void ở ví dụ trên, kiểu trả về là giá trị chúng ta nhận được sau khi gọi hàm.

Bạn hãy nhớ lại, khi truyền x vào f(x) chúng ta sẽ nhận lại là y. Thì hàm cũng vậy, chúng ta có thể trả lại một giá trị gì đó. ví dụ:

// [kiểu trả về]: int

public static int tong(int x, int y){

int t = x + y; // Tính tổng 2 só x, y

return t; // trả số đó ra sử dụng câu lệnh `return {biến}`

}

public static void main(String[] args){

int t = tong(5,6); // Lấy giá trị trả ra, gán nó vào t;

}

Tôi định nghĩa một hàm tính tổng tong(x,y) nhận vào 2 số nguyên, và yêu cầu nó trả ra một số int.

Các kiểu trả về:

* primitive: int, boolean, char, ...
* Object: String, (còn rất nhiều, sẽ học ở bài tiếp theo)
* void: Không trả về gì cả

Ở ví dụ đầu tiên mình đã sử dụng void để định nghĩa hàm.

public static void f(int x, int y){

int a = x + y;

System.out.println("In a ra màn hình: " + a);

}

Điều này nói là hàm của chúng ta thực hiện một hoạt động khép kín, và không có nhu cầu trả ra ngoài cái gì cả. Mình chỉ tính tổng rồi in luôn ra màn hình thôi, không cần đưa gì ra ngoài cả.

3 - [danh\_sách\_tham\_số]

Tham số đầu vào, là những thứ chúng ta đưa vào hàm, định nghĩa tham số đầu vào bao gồm [kiểu\_dữ\_liệu] [tên\_biến]. Chúng ta có truyền nhiều tham số vào hàm bằng cách đặt dầu phẩy , giữa mỗi tham số.

public static int f(int x, int y, int z, ... ){

// code

}

Ở đây lưu ý phần [tên\_biến] bạn có thể đặt tên bất kỳ. chẳng hạn:

// Hàm nhận vào 2 biến `x`, `y` và trả ra kết quả `boolean` xem nó có bằng nhau hay không

public static boolean bangnhau(int x, int y){

return x == y;

}

public static void main(String[] args){

int a = 5; // tên biến là `a`

int b = 6; // tên biến là `b`

boolean ketqua = bangnhau(a,b); // đưa `a` , `b` vào hàm.

// bản chất khi gọi hàm `bangnhau`:

// int x = a;

// int y = b;

// return x == y;

//

System.out.println("Kết quả: " + ketqua);

}

Bạn định nghĩa tham số đầu vào là x và y thì nó chỉ hiểu trong ở hàm đó thôi, và những giá trị truyền vào sẽ gán vào các biến x và y.

**Thực hành**

Sau đây chúng ta sẽ ứng dụng những kiến thức đã học để làm một chương trình đơn giản:

Cho 3 số nguyên a, b, c tượng trưng cho 3 cạnh AB, AC, BC của tam giác ABC. Kiểm tra ABC có phải tam giác không? nếu có, là tam giác gì?

**Nhắc lại**: Chúng ta chưa học **nhập xuất dữ lịệu** trong Java nên chúng ta sẽ gán giá trị ngay trong code và chạy test.

public class Calculation {

public static void main(String[] args) {

int a = 5, b = 5, c = 5; // tường hợp tác giác đều

// Các bạn tự thay số vào để kiểm tra nhé.

if (laTamgiac(a,b,c)){

System.out.println("Là tam giác!");

if(laTamgiacDeu(a,b,c)){

System.out.println("Và còn đều nữa!");

// Là tam giác đều thì không cần kiếm tra điều kiện còn lại nữa.

}else {

if (laTamgiacVuong(a, b, c)) {

System.out.println("Và còn vuông nữa!");

// Không thể xảy ra vuông cân. Vì chúng ta đầu vào chỉ là số nguyên.

// Còn muốn đầy đủ, bạn phải kiểm tra trường hợp vừa vuông vừa cân nữa.

}

if (laTamgiacCan(a, b, c)) {

System.out.println("Và còn cân nữa!");

}

}

}else{

System.out.println("Không phải là tam giác!");

}

}

public static boolean laTamgiac(int a, int b, int c) {

if ((a + b) > c && (a + c) > b && (b + c) > a) {

// Tam giacs: tổng 2 cạnh phải lớn hơn cạnh còn lại

return true;

} else {

return false;

}

}

public static boolean laTamgiacVuong(int a, int b, int c){

if ((a\*a + b\*b) == c\*c || (a\*a + c\*c) == b\*b || (b\*b + c\*c) == a\*a) {

// Là tam giác vuông nếu có 1 trong các đièu kiện thoả mãn pythagore.

return true;

} else {

return false;

}

}

public static boolean laTamgiacCan(int a, int b, int c){

if (a == b || b == c || c == a) {

return true;

}else{

return false;

}

}

public static boolean laTamgiacDeu(int a, int b, int c){

if (a == b && b == c) {

return true;

}else{

return false;

}

}

}