Khác biệt giữa Primitive datatype và refercen Datatype

Contents

[Default value and Null 1](#_Toc70124574)

[Lưu trữ chúng 1](#_Toc70124575)

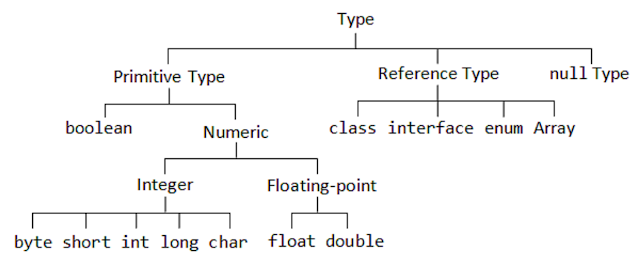
[Sử dụng toán tử = operator 1](#_Toc70124576)

[So sánh sử dụng == 2](#_Toc70124577)

[Return Giá trị của hàm 4](#_Toc70124578)

[Độ lớn lưu trư 4](#_Toc70124579)

[Tham Khảo 4](#_Toc70124580)



# Default value and Null

**Primivate** sẽ được tự động khởi tạo giá trị default nếu không có chỉ định nó.

**Reference** giá trị khởi tạo mặc định là Null, sẽ bị nullpointer nếu mà truy cập nó khi chưa khởi tạo

# Lưu trữ chúng

Primitive sẽ lưu trữ giá trị của nó trong bộ nhớ stack

Reference sẽ handle một giá trị trong bộ nhơ head , nó không phải pointer như c++ vì nó có thể được truy cập và thay đổi trạng thái object của nó.

# Sử dụng toán tử = operator

Primitive thực sự được copy value và gán cho giá trị mới cho đối được nhận, 2 bộ nhớ khác nhau, không ảnh hưởng đến nhau khi 1 trong 2 thay đổi giá trị.

Reference chỉ là handle copy đối được chỉ được shared cho cả 2 , nếu 1 trong 2 thay đổi đối tượng shared thì nó sẽ ảnh hưởng đến giá trị của các Reference type đang tham chiếu.

int i = 20;

int j = i;

j++; *// will not affect i, j will be 21 but i will still be 20*

System.out.printf("value of i and j after modification i: %d, j: %d %n", i, j);

List<String> list = new ArrayList(2);

List<String> copy = list;

copy.add("EUR"); *// adding a new element into list, it would be visible to both list and copy*

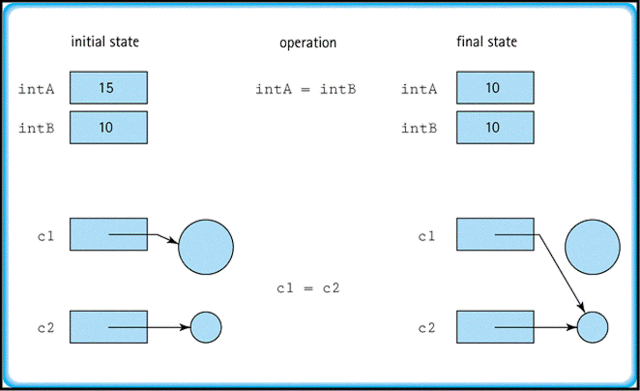
System.out.printf("value of list and copy after modification list: %s, copy: %s %n", list, copy);

Output :

value of i and j after modification i: 20, j: 21

value of list and copy after modification list: [EUR], copy: [EUR]

Xem thêm hình minh họa dưới:



# So sánh sử dụng ==

Primitive so sánh thực sự giá trị mà nó đang chứa

Reference chỉ so sánh xem coi cả 2 có chung vùng nhớ (địa chỉ) xem nếu chung thì bằng

int i = 20;

int j = 20;

if (i == j) {

System.out.println("i and j are equal");

}

String JPY = new String("JPY");

String YEN = new String("JPY");

if (JPY == YEN) {

System.out.println("JPY and YEN are same");

}

if (JPY.equals(YEN)) {

System.out.println("JPY and YEN are equal by equals()");

}

Output :

i and j are equal

JPY and YEN are equal by equals()

Khi so sánh 2 đối tượng reference thì nên sử dụng equal để so sánh.

Truyền Primitive và Reference vào method ý là paramater đó

Primitive **không** bị thay đổi giá trị ban đầu khi thay đổi giá trị của nó trong hàm.

Reference **Sẽ bị** thay đổi giá trị ban đầu khi thay đổi giá trị của nó trong hàm :

public class PrimitiveVsReference{

private static class **Counter** {

private int count;

public void advance(int number) {count += number;}

public int getCount() {return count;}

}

public static void main(String args[]) {

**int i** = 30;

System.out.println("value of i before passing to method : " + i);

print(30);

System.out.println("value of i after passing to method : " + i);

Counter myCounter = new Counter();

System.out.println("counter before passing to method : " + myCounter.getCount());

print(myCounter);

System.out.println("counter after passing to method : " + myCounter.getCount());

}

public static void print(Counter ctr) { ctr.advance(2); }

public static void print(int value) { value++; }

}

Output :

value of i before passing to method : 30

value of i after passing to method : 30

counter before passing to method : 0

counter after passing to method : 2

**Có thể thấy được đối Counter bị thay đổi giá trị khi vào hàm , nhưng int i không bị thay đổi khi vào hàm**

# Return Giá trị của hàm

**Primitive return** giá trị thực sự .

Reference sẽ trả về handle object lưu trong head.

# Độ lớn lưu trư

Reference sữ tốn nhiều bộ nhớ vì phải lưu thêm các metadata, header ...

# Tham Khảo

https://javarevisited.blogspot.com/2015/09/difference-between-primitive-and-reference-variable-java.html#axzz6ssyo0TAv