Parameter Passing Techniques in Java

→ Parameters refers to the list of variables ~~of~~ in a method declaration.

→ Arguments are the actual values that are passed in when the method is invoked. When we invoke a method, the arguments used must match the declaration's parameters in type & order.

→ We can use any data type for a parameter of a method including int, float, double, boolean, char (primitives types) and reference data types such as object & arrays, Strings.

e.g:- 1

```
public class Demo {
    static void updateVariable (int a) {
        a = 10;
    }
    public static void main (String[] args) {
        int num = 5;
        Updatevariable (num);
        System.out.println (num);
    }
```

⇒ example 1

Output? :- 5 [value is not being updated]

eg 2 Now in above example if we write:-

```
public class Demo {
    int num;
    static void updateVariable (Demo d1) {
        d1.num = 10;
    }
psvm
{
    Demo d = new Demo();
    d.num = 5;
    updateVariable (d);  SOP(d.num); }
```

example 2 ⇒

d1 is simply a reference to the existing object passed from the caller. We don't need new here as we are not creating a new object in the parameter list

⇒ output:- 10 ↓ updated value

In first example we were passing primitive type (integer) and in second we were passing object reference.

There are basically 2 techniques to pass parameters :-

1. Pass by Value (Call by Value)
2. Pass by Reference (Call by Reference)

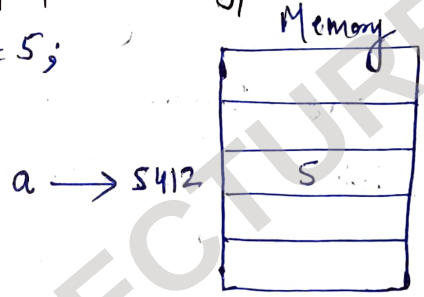
} Both are technically only Pass-by Value

NOTE:- Java uses Pass by Value

How a simple variable of primitive type stored in memory:-

int a = 5;

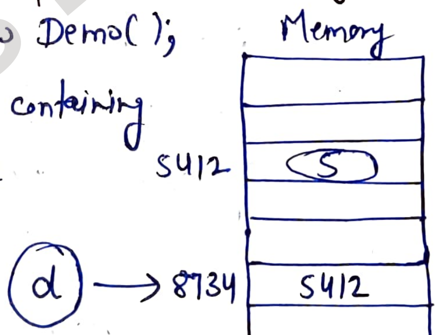
at runtime we have only memory address & the data (value)



How an object reference represented in memory:-

Demo d = new Demo( );

d is an object reference containing the address of where the actual value is stored.



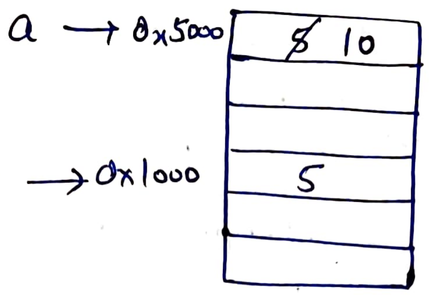
1. Pass by Value:- In Java all arguments are passed by Value. It means when we pass a variable to a method, Java passes a copy of the variable's value to the method.

⇒ Pass-by Value for primitives:- (int, float, double etc.)

↳ Copy of actual value is to be passed. Hence any change made to the parameter inside the method do not affect the original variable outside the method.

Let's illustrate example 1.

→ So here num & a both variables are stored at separate memory locations (0x5000 & 0x1000), So changes to a do not affect the num.

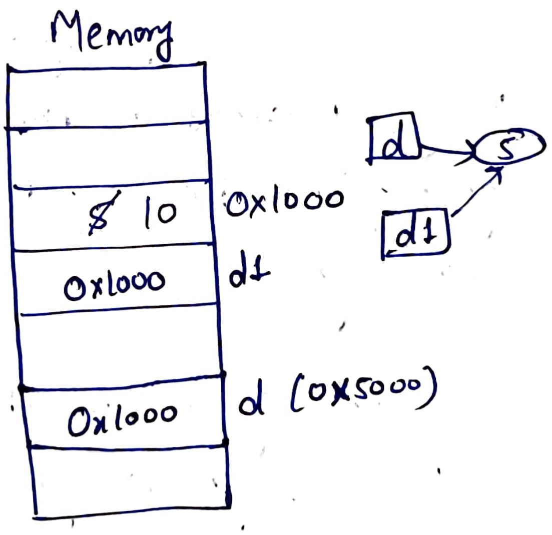


Java copies only the value of num into a. The original memory location remains untouched.

⇒ $\boxed{\text{Pass by Value for Object References!-}}$ When we pass an object to a method, then java passes copy of the reference to the object. not the actual object itself.

It may appear like pass-by-reference but it's still Pass-by-value because java is copying the reference value (i.e. the address of the object)

Let's illustrate example 2.



Here both the object, points to same memory location 0x1000
Hence it will affect the original object state.

Here both d & d1 referring to the same ~~[illegible]~~ object.

Demo

example 3 :-

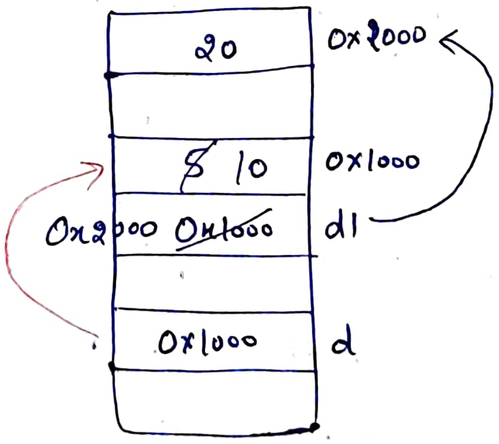
```
public class Demo {
        int num;
    static void updateVariable (Demo d1) {

            d1.num = 10;
            d1 = new Demo();    // Reassignment
            d1.num = 20;
    }
    public static void main (String [] args) {
          Demo d = new Demo();
           d.num = 5;
           UpdateVariable (d);
          System.out.println (d.num);
    }
```

newobject is created & d1 now references it

it only affects the local object reference

Output :- 10



IMP NOTE

→ Sometimes we call pass by value for object References as Pass by Reference But it is Pass-by value only.

IMP

→ So technically in Java there is no true Pass-by-Reference. There is only Pass-by-value always.

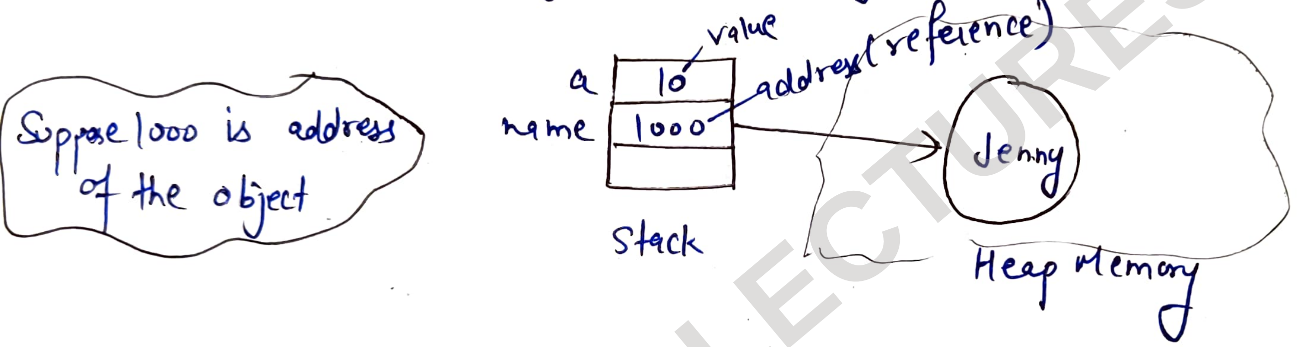
Stack & Heap Memory:-

→ When we declare or instantiate an object, the actual object goes on heap and the address (reference) of the object goes on stack.

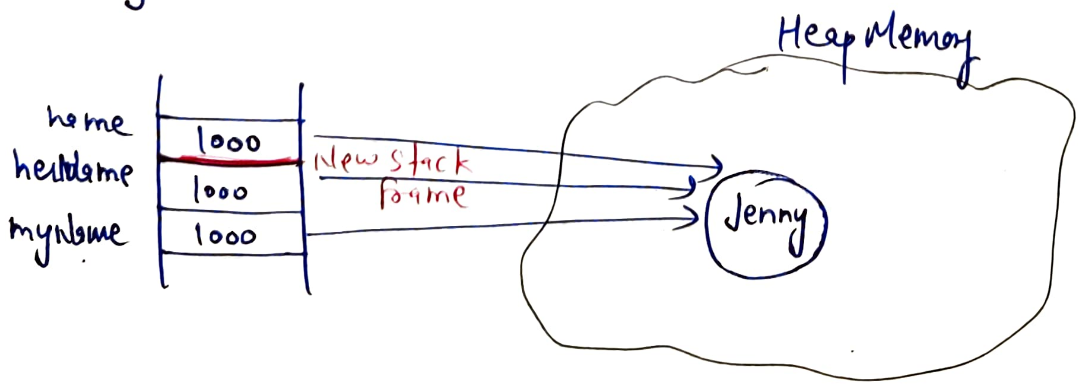
→ Stack Memory is used for method calls & local variables within methods

→ In Java Stack & heap are two distinct memory areas that serve different purposes & have different characteristics.

e.g:-
```
int a = 10;
String name = "Jenny";
```



e.g:-
```
static void display(String name){
    System.out.println(name);
}
public static void main(String[] args) {
    String myName = "Jenny";
    String herName = myName;
    display(herName);
}
```


Heap Memory
name
herName
myName
1000
1000
1000
New stack frame
Jenny