**Parameter Passing in Java**

***Pass-by-value:***

                A copy of the passed-in variable is copied into the argument of the method. Any changes to the argument do not affect the original one.

***Pass-by-reference:***

The argument is an alias of the passed-in variable. Any changes to the argument will affect the original one.

Parameters of a method are specified in the method declaration. When we call a method, we must provide values that match with the types of the parameters. In the following example, the sum method accepts two integers and returns their summation.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | public static int sum(int a, int b)  {      return a + b;  }  public static void main(String[] args)  {      int c = sum(1, 2);         // c is 3      int x = 3, y = 4;      int z = sum(x, y);         // z is 7  } |

Parameter variables work as local variables in the method. They are initialized by the values passed from the calling code. But note that, parameter variables are different from the variables in the calling code. This approach is known as *pass by value*. (In some other languages, there is an alternative approach known as *pass by reference*, in which parameter variables could be the same variables from the calling code.)

In the next example, the parameter variable a in dummyIncrease is only visible inside the method. Although it is increased by 1, this will affect the variables x or a in the main method; they are difference variables with different memory locations. Note that the parameter variable a in dummyIncrease and the local variable a in main are two different variables.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public static void dummyIncrease(int a)  {      a++;  }  public static void main(String[] args)  {      int x = 3;      dummyIncrease(x);          // x is still 3      int a = 4;      dummyIncrease(a);          // a is still 4  } |

Object variables in parameters are more complicated than primitive variables. Since object variables are essentially references to objects, the values passed to object variables in parameters are references (not objects themselves). In the next example, dummyIncreaseaccepts two parameters: one is an array, and the other is an integer. Since arrays are objects, the parameter variable x refers to the same array as the variable arr from the calling code. Thus, the increase of the first element of the array is still effective after the method returns.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public static void dummyIncrease(int[] x, int y)  {      x[0]++;      y++;  }  public static void main(String[] args)  {      int[] arr = {3, 4, 5};      int b = 1;      dummyIncrease(arr, b);      // arr[0] is 4, but b is still 1  } |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | main()    arr +---+       +---+---+---+        | # | ----> | 3 | 4 | 5 |        +---+       +---+---+---+    b   +---+             ^        | 1 |             |        +---+             |                          |  dummyIncrease()         |    x   +---+             |        | # | ------------+        +---+    y   +---+        | 1 |        +---+ |

As another example, the following code defines a method swap which exchanges two elements of an array.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | public static void swap(int[] arr, int i, int j)  {      int tmp = arr[i];      arr[i] = arr[j];      arr[j] = tmp;  }  public static void main(String[] args)  {      int[] arr = {3, 4, 5, 6};      swap(arr, 1, 3);      // arr becomes {3, 6, 5, 4}  } |

**public** **class** PassByReferenceConfusion {  
    
    **public** **static** **void** main(**String** args[]) {  
       Car car = **new** Car("BMW");  
       **System**.out.println("Brand of Car Inside main() before: "+ car.brand);  
       printBrand(car);  
       **System**.out.println("Brand of Car Inside main()after: "+ car.brand);  
    }  
    
    **public** **static** **void** printBrand(Car car){  
        car.brand = "Maruti";  
        **System**.out.println("Brand of Car Inside printBrand(): "+car.brand);  
    }  
    
    **private** **static** **class** Car{  
        **private** **String** brand;  
        
        **public** Car(**String** brand){  
            **this**.brand = brand;  
        }  
  
    }  
}  
  
**Output:**  
Brand of Car Inside main() before: BMW  
Brand of Car Inside printBrand(): Maruti  
Brand of Car Inside main()after: Maruti