**Passing Parameters in Java**

In Java, **everything is pass-by-value**. However, the key is understanding what is being passed.

**Primitive Data Types**

When you pass a primitive type (like int, double, char, etc.) to a method, **a copy of the value** is passed. Changes to the parameter inside the method do not affect the original variable.

Example:

public class Main {

public static void changePrimitive(int num) {

num = 10; // This change is local to the method

}

public static void main(String[] args) {

int original = 5;

changePrimitive(original);

System.out.println(original); // Output: 5

}

}

**Objects and Complex Data Types**

When you pass an object or a reference variable (like String, ArrayList, or any custom object), **the value of the reference to the object** is passed. This means both the original reference and the parameter reference point to the same object in memory.

* If you modify the **contents of the object** through the reference, those changes will reflect in the original object.
* If you reassign the reference to a new object, the original reference remains unchanged.

Example 1: Modifying the contents of an object

import java.util.ArrayList;

public class Main {

public static void changeList(ArrayList<String> list) {

list.add("New Item"); // Modifying the object

}

public static void main(String[] args) {

ArrayList<String> myList = new ArrayList<>();

myList.add("Original Item");

changeList(myList);

System.out.println(myList); // Output: [Original Item, New Item]

}

}

Example 2: Reassigning the reference

public class Main {

public static void changeReference(StringBuilder sb) {

sb = new StringBuilder("New Object"); // Reassigning reference

sb.append(" Modified");

}

public static void main(String[] args) {

StringBuilder original = new StringBuilder("Original");

changeReference(original);

System.out.println(original); // Output: Original

}

}

**Key Takeaways:**

1. **Primitive types:** A copy of the value is passed. Changes inside the method do not affect the original variable.
2. **Objects:** A copy of the reference is passed. Changes to the object through the reference are visible to the original.
3. **Reassigning the reference inside the method:** The reassignment is local to the method and does not affect the original reference.