In Java, the terms "pass by value" and "pass by reference" refer to how arguments are passed to methods.

Pass by Value:

Java uses "pass by value" for all method calls. When you pass a variable as an argument to a method, a copy of the variable's value is made and passed to the method. Any changes made to the parameter within the method do not affect the original variable in the calling code.
Here's an example to illustrate pass by value:

```
public class PassByValueExample {

    public static void main(String[] args) {
        int number = 10;
        System.out.println("Before method call: " + number);
        modifyValue(number);
        System.out.println("After method call: " + number);
    }

    public static void modifyValue(int value) {
        value = 20;
        System.out.println("Inside method: " + value);
    }
}
```

In the above example, the modifyValue method receives a copy of the number variable. When the method assigns a new value to the value parameter, it does not affect the original number variable. The output will be:

```
Before method call: 10
Inside method: 20
After method call: 10
```

Pass by Reference:

Unlike some programming languages, Java does not have explicit pass-by-reference. In Java, object references are passed by value. When you pass an object as an argument to a method, a copy of the reference is made and passed to the method. This means you can modify the object's state within the method, but you cannot reassign the reference to point to a different object.
Here's an example to demonstrate pass by reference (or rather, pass by value with object references):

```
public class PassByReferenceExample {

    public static void main(String[] args) {
        StringBuilder name = new StringBuilder("John");
        System.out.println("Before method call: " + name);
        modifyReference(name);
        System.out.println("After method call: " + name);
    }

    public static void modifyReference(StringBuilder value) {
        value.append(" Doe");
        System.out.println("Inside method: " + value);
    }
}
```

In the above example, the modifyReference method receives a copy of the reference to the name object. The method can modify the state of the StringBuilder object by appending " Doe" to it. The output will be:

```
Before method call: John
Inside method: John Doe
After method call: John Doe
```

Note that even though the object's state changed, the original reference name still points to the same object.

```
package com.hdfc.var.args;

import java.util.ArrayList;
import java.util.List;

public class PassByValuePassByReference {

    //pass by value
    //primitive, String, Wrapper

    //pass by reference
    //List, StringBuilder, StringBuffer

    public static void main(String[] args) {
        String a = "hello";
        //Integer
        int a2 = 10;
        test(a2);//Integer
        System.out.println(a);
        test(a);
        System.out.println(a);
    }

    static void  test(Object a){
        int i = Integer.parseInt((String) a);
    }
}
```