Consider the following class MyInteger

```java
public class MyInteger {
    private int i;

    public MyInteger(int j) {
        i = j;
    }

    public int getI() {
        return i;
    }

    public void setI(int j) {
        i = j;
    }

    public void change(int x) {
        x = x + 1;
    }

    public void changeInt(MyInteger another) {
        int j = another.i;
        another.setI(j + 1);
    }
}
```

Consider the following program that uses the class MyInteger

```java
public class IntegerTest {
    public static void main(String[] args) {
```
MyInteger m = new MyInteger(3);

int z = 8;
System.out.println(z);
m.change(z);
System.out.println(z);

What is the output? It is 8 and 8, not 8 and 9. To see this, we must understand the internals of parameter passing. In the above program, we are calling the method change. The method change looks like:

```
public void change(int x) {
    x = x+1;
}
```

Here x is called formal parameter. In the main method, we calling the method as m.change(z). We are calling the method with parameter z. Here z is called actual parameter. Whenever a method is called, the following actions will happen.

1. Reserve a block of memory for the method.
2. Create memory locations for each of the formal parameters (these will be created in the block of memory that is reserved in first step).
3. For each of the formal parameters, perform the assignment statement:

   \[ formal\text{parameter} = \text{actualparameter} \]

4. Perform the body of the method.
5. Erase the memory reserved for the method (and so the memory for the formal parameters).

Now lets understand what happens when call the method m.change(z). Before we call the method, the value of z is 8. So the first print statement will output 8. When m.change(z) is called, the following will happen.

1. Reserve a block of memory for method change
2. Create a memory location named x (Because x is the formal parameter of change)
3. Do the assignment
Now, the value $x$ becomes 8.

4. Perform the body of the method. This will set the value of $x$ to 9. Notice that $z$ is unchanged.

5. Erase memory reserved for the method `change`

When the execution of the method `change` is over, the value of $z$ is still 8. So the second print statement will also output 8.

Now consider the following program.

```java
public class IntegerTest {
    public static void main(String[] args) {
        MyInteger m = new MyInteger(3);
        MyInteger a = new MyInteger(5);

        System.out.println(a.getI());
        m.changeObj(a);
        System.out.println(a.getI());
    }
}
```

What is the output? Now it is 5 and 6. To see this, lets trace the execution of the method. The method `changeObj(MyInteger another)` has formal parameter named `another`. In the main method it is being called as `m.changeObj(a)`. So the actual parameter is `a`. What are the contents of the memory location `a`? It is the address an object. It points to a `MyInteger` object and in this object the value of `$i$` is 5. When the method `m.changeObj(a)` is called, the following happens:

1. Reserve a memory location for the method `changeObj`

2. The formal parameter is named `another`. So create a memory location with name `another`.

3. Perform the assignment statement

   ```java
   another = a
   ```

   This will copy contents of `a` into `another`. As noted before, contents of `a` is the address of a `Myinteger` object (in which `$i$` is 5). Thus both `another` and `a` are pointing to the same object.
4. Now perform the body of the method. The body has following two statement.

```java
int j = another.i;
another.setI(j+1);
```

First statement will make \( j \) equal 5. Now, it will perform `another.setI(6)`. This will call the method `setI(6)` in the object pointed by `another`. Note that both `a` and `another` are pointing to the same object. Thus in this object the value of `i` becomes 6.

5. This is the end of the method. So erase the memory reserved for the method `changeObj`.

Now look at the object pointed by `a`. In this object, the value of `i` is 6. Thus the second print statement will output 6.