Names Don't Matter

Understanding Variables, Scope, and Pass by Value

The Big Idea

Variable names are arbitrary labels for HUMANS

The computer doesn't care if you call it:

- x
- myNumber
- banana
- qwerty1345

As long as you use it consistently, it all works the same!

What Actually Matters to the Computer

Methods care about:

1. Position (which parameter is first, second, etc.)

2. Type (int, String, double, etc.)

3. Data flow (assignments, returns)

Methods DO NOT care about variable names!

Same Code, Different Names

```java
int x = 5;
int y = 10;
int result = add(x, y);

public static int add(
    int a, int b) {
  return a + b;
}
```

```java
int banana = 5;
int apple = 10;
int sum = add(banana, apple);

public static int add(
    int q, int r) {
  return q + r;
}
```

These do EXACTLY the same thing!
The names changed, but the behavior is identical.

Position Matters, Names Don't

```
public static int subtract(int first, int second) {
    return first - second;
}
```

```
int a = 10;
int b = 3;
subtract(a, b);
// Returns 7
```

```
int a = 10;
int b = 3;
subtract(b, a);
// Returns -7
```

Same variable names, different ORDER = different result!

Scope: Each Method Has Its Own Names

Variables live in the block where they're declared

```
public static void main(String[] args) {
    int x = 5;  // This x lives in main
    foo();
}

public static void foo() {
    int x = 10;  // This x lives in foo
    System.out.println(x);  // Prints 10
}
```

These are TWO DIFFERENT variables that happen to have the same name!

Scope: Names Are Local Labels

```
public static void main(
    String[] args) {

    int x = 5;

    System.out.println(x);
    // Prints 5

    foo();

    System.out.println(x);
    // Still prints 5!
}
```

```
public static void foo() {

    int x = 10;

    System.out.println(x);
    // Prints 10

    x = 20;

}
```

The x in main and the x in foo are completely separate!
Changing one doesn't affect the other.

Pass by Value: Copying Data

When you call a method, Java COPIES the value into the parameter

Think of it like:

- You write a number on a piece of paper
- You make a photocopy
- You give the copy to someone else
- They can write on THEIR copy
- Your original is unchanged!

Pass by Value in Action

```
public static void main(String[] args) {
    int num = 5;
    tryToChange(num);
    System.out.println(num);  // Still prints 5!
}

public static void tryToChange(int x) {
    x = 100;  // This only changes the COPY
    System.out.println(x);  // Prints 100
}
```

The parameter x gets a COPY of num's value.
Changing x doesn't change num!

Names Don't Matter: Pass by Value

Whether the parameter has the same name or different name doesn't matter!

```
// Same name
int x = 5;
foo(x);
System.out.println(x);
// Prints 5

void foo(int x) {
    x = 10;
}
```

```
// Different name
int x = 5;
foo(x);
System.out.println(x);
// Prints 5

void foo(int banana) {
    banana = 10;
}
```

SAME RESULT! The name of the parameter doesn't change the fact that it's a separate copy.

Practice Problem

```
public static void main(String[] args) {
    int x = 1;
    int y = 0;
    System.out.println(x + " ");
    System.out.println(y + " ");
    a(x);
    y = b(x);
    System.out.println(x + " ");
    System.out.println(y + " ");
}
public static void a(int x) {
    x = x + 1;
}
public static int b(int x) {
    return x + 1;
}
```

What does this print? What are x and y after each line?

Tracing Table

Track variables after each line - names don't matter, values do!

After this line...	x (main)	y (main)	Output
int x = 1;	1		
int y = 0;	1	0	
println(x + " ");	1	0	1
println(y + " ");	1	0	1 0
a(x);	1	0	1 0
y = b(x);	1	2	1 0

Why a(x) Didn't Change x

```
a(x);  // x is still 1 after this!

public static void a(int x) {
    x = x + 1;  // This changes the COPY
}
```

- Method a gets a COPY of x's value
- The parameter (also called x) is a different variable
- Changing the parameter doesn't affect main's x
- Pass by value means changes don't go back!

Why y = b(x) Changed y

```
y = b(x);   // NOW y becomes 2!

public static int b(int x) {
    return x + 1;   // Returns a value
}
```

- Method b gets a COPY of x's value (1)
- It returns a NEW value (2)
- The assignment y = stores that return value
- THIS is how methods change variables: through return values and assignment!

Same Behavior, Ridiculous Names

This does EXACTLY the same thing as the previous example!

```
public static void main(String[] args) {
    int banana = 1;
    int qwerty1345 = 0;
    a(banana);
    qwerty1345 = b(banana);
    System.out.println(banana);        // 1
    System.out.println(qwerty1345);   // 2
}

public static void a(int potato) {
    potato = potato + 1;
}

public static int b(int carrot) {
    return carrot + 1;
}
```

Names don't matter! Position, type, and data flow matter!

Key Takeaways

1. Variable names are for HUMANS
   Computers only care about position, type, and data flow

2. Scope means each method has its own variables
   Same name in different methods = different variables

3. Pass by value means methods get COPIES
   Changing parameters doesn't affect original variables

4. To change a variable, use return value and assignment.
   Assignment is how data flows back to the caller

Names Don't Matter!

You could rename every variable in your program and it would work exactly the same way.

What matters is:
✓ Types match
✓ Positions are correct
✓ Data flows through assignments and returns

Master these concepts and you'll understand how methods really work!