AP CSA Method and Scope

Xuhui Liu

Method and scope

test.java

```
1 class test {
2   public static void main(String args[]) {
3     System.out.println(square(3));
4   }
5
6   static int square(int x) {
7     return x*x;
8   }
9 }
```

call your method

return type

method name

parameter

method ends when return is encountered !

a method has the following structure

(keyword) returntype methodname

(parameter) { }

Scope

```
test.java:6: error: cannot find symbol
        System.out.print(i);
                         ^
  symbol:   variable i
  location: class test
```

st.java

```
class test {
  public static void main(String args[]) {
    for (int i = 0; i < 3; i++) {
      System.out.print(i + " ");
    }
    System.out.print(i);
  }
```

★ The variable inside a block "{}" cannot be accessed outside the block

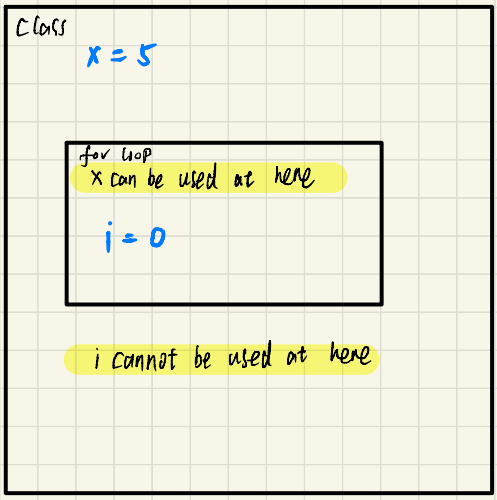
Scope

```
5 5 5
```

test.java

```
1 class test {
2   public static void main(String args[]) {
3     int x = 5;
4     for (int i = 0; i < 3; i++) {
5       System.out.print(x + " ");
6     }
7   }
8
```

The variable outside the block can be used inside the block



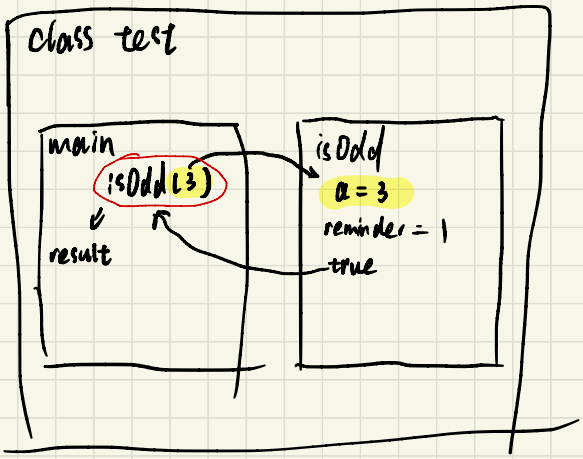
Method is also a block, so same rule applies for method.

Method parameters and return values

test.java

```
1 class test {
2   public static void main(String args[]) {
3     boolean result = isOdd(3)
4     System.out.println(result);
5   }
6
7   static int square(int x) {
8     return x*x;
9   }
10          return type  name  parameter
11  static boolean isOdd(int a) {
12    int reminder = a%2;
13    if (reminder == 0) {
14      return false;
15    }
16    else {
17      return true;
18    }
19  }
20 }
```

What is happening behind the scene



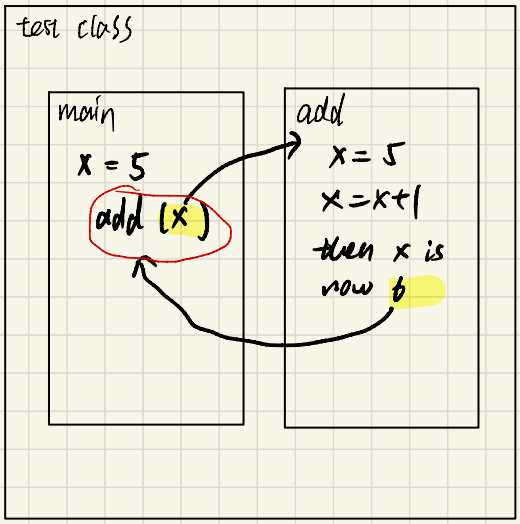
you basically make a copy of parameters in the method when the input is primitive type ⇒

```
5
```

test.java

```
1 class test {
2   public static void main(String args[]) {
3     int x = 5;
4     add(x);
5     System.out.println(x);
6   }
7
8   static int add(int x) {
9     x = x + 1;
10    return x;
11  }
12
```

return type name parameter



Note: "x" in main is still 5.
So, method cannot modify its primitive type parameter

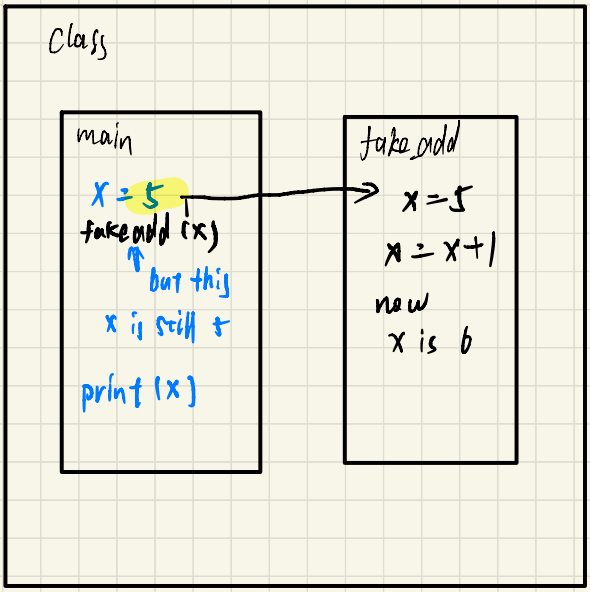
Void as return type

```
5
```

test.java

```
1 class test {
2   public static void main(String args[]) {
3     int x = 5;
4     fake_add(x);
5     System.out.print(x);
6   }
7
8   static void fake_add(int x) {
9     x = x+1;
10  }
11
```

no return statement in void

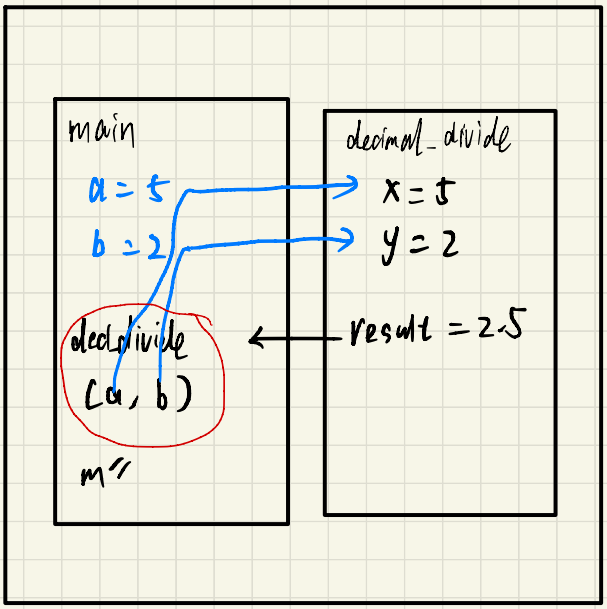


Method with multiple parameters

```
2.5
```

test.java

```
1 class test {
2   public static void main(String args[]) {
3     int a = 5;
4     int b = 2;
5     double m = decimal_divide(a, b);
6     System.out.print(m);
7   }
8
9   static double decimal_divide(int x, int y) {
0     // assume y is not zero
1     double result = ((double) x)/y;
2     return result;
3   }
```

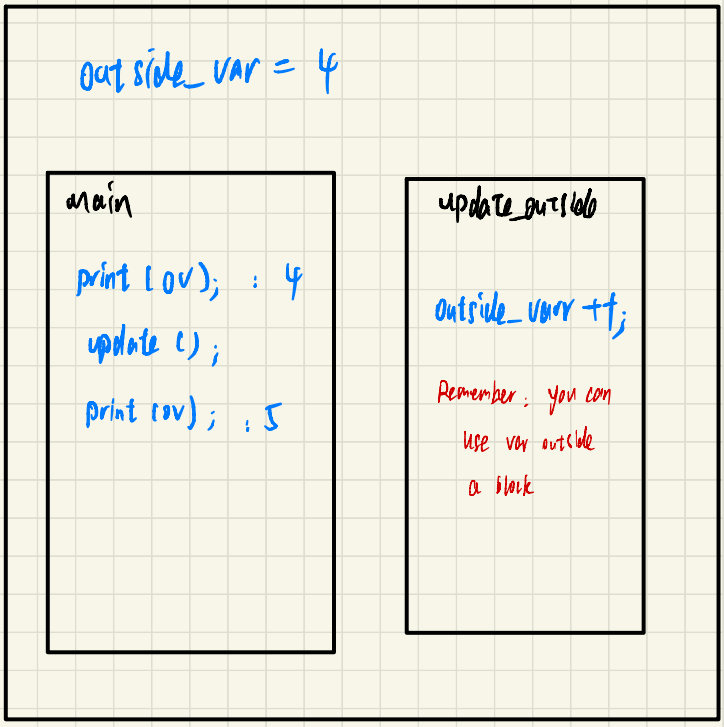


Method can change value of variables outside its block

```
4
5
```

test.java

```
1 class test {
2
3   static int outside_var = 4;
4
5   public static void main(String args[]) {
6     System.out.println(outside_var);
7     update_outside();
8     System.out.println(outside_var);
9   }
0
1   static void update_outside() {
2     outside_var++;
3   }
```



总结：

当你调用一个方法时，你在把他的 parameter 在 method 的 scope 里 make 了一个 copy. 此时方法内部的任何事都不会影响到这些 params 外面的值. 只有你方法 return 的东西可以被送到外面.