

Introduction to Programming

with Java, for Beginners

Debugging Tips Scope

Debugging

- It is highly unlikely that you will write code that will work on the first go
- Bugs or errors
 - Syntax
 - Fixable if you learn to read compiler error messages
 - Semantic
 - No easy fix
 - Use print statements to our advantage

1

Syntax Error

- Use the Dr Java tool to your advantage
 - Keywords turn blue
 - Comments turn green
 - { } matching
- Reading compiler Errors
 - Turn on line numbers
 - In DrJava (Edit Preferences -> Display)
 - Learn common syntax errors

2

Debugging with System.out.println

```
public static int sumOdd(int n){
    //sum positive odd numbers upto n
    // e.g. sum(3) = 4
    int sum = 0;
    for (int i = 1; i <= n; i = i + 1){
        sum = sum + i;
        System.out.println(i + ":" sum);
    }
    return sum;
}
```

Result of print
1 :1
2 :3
3 :6

Remember to comment out print statement when you are done testing

3


DRY Principle

- Using Methods within other Methods
- If *method1* is being used by another *method2*
 - If method1 is within the same class as method2, then simply call it:
method1(parameter);
 - If method1 is in different class than method2, then to call it do:
className.method1(parameters);
 - This works only if method1 was made *public* in its method header

4

Example DRY Principle

```
public static int sumOdd(int n){
    //sum positive odd numbers upto n
    // e.g. sum(3) = 4
    int sum = 0;
    for (int i = 1; i <= n; i = i + 1){
        if(isOdd(i)) {
            sum = sum + i;
        }
    }
    return sum;
}
```



5

Scope

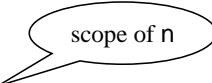
- *Scope* means the area of code in which an entity is known (or alive)
 - Mainly concerned with *variables* and *methods*
 - Which parts of the program can access them?
- Sometimes scope is *explicitly* designated with a keyword
 - *private*: known only within the class
 - *public*: known outside of (and within) the class
 - Note that Methods have explicit scope
- Other times it is *implicitly* designated by location

6

Implicit Scope: Method Parameters

- A method parameter is an “input variable”
- *Scope*: the method in which it is defined
- No other method can access (read/write) it

```
public static int absoluteValue(int n){
    if (n < 0) {
        return -n;
    }
    else {
        return n;
    }
}
```



7

Implicit Scope: Local Variables

- A “local variable” is defined within a method body { }
- They are inherently private to the method in which they are defined
- We don't use public/private for local variables
- It may be defined in a block { } within a method body
- **Scope:** point of declaration to end of closest enclosing block

8

Example of Local Variables

```
public static int isLarger(int x, int y){  
    if (x > y) {  
        int larger = x;  
    }  
    else {  
        int larger = y;  
    }  
    return larger;  
}
```

scope of larger

scope of a different larger

Illegal: not declared in current scope

9

Another Example

```
int fibonacci(int limit) {  
    int first = 1;  
    int second = 1;  
    while (first < 1000) {  
        System.out.print(first + " ");  
        int next = first + second;  
        first = second;  
        second = next;    next  
    }  
    System.out.println();  
    second first limit  
}
```

10

For Loop Special Case

- The **for** loop is a special case
- You can declare variables in the **for** statement
- The scope of those variables is the entire **for** loop
- This is true even if the loop is not a block i.e. { }

```
void multiplicationTable() {  
    for (int i = 1; i <= 10; i++) {  
        for (int j = 1; j <= 10; j++)  
            System.out.print(" " + i * j);  
        System.out.println();  
    }  
}
```

11