

Lecture Jan 24 MW - Flow and Logic

Bentley James Oakes

January 23, 2018

- If you want extra programming problems, check out <http://codingbat.com/java>

Java

Python

Warmup-1 > intMax

[prev](#) | [next](#) | [chance](#)

Given three int values, a b c, return the largest.

intMax(1, 2, 3) → 3
intMax(1, 3, 2) → 3
intMax(3, 2, 1) → 3

Go

...Save, Compile, Run

Show Solution

```

public int intMax(int a, int b, int c) {
    return a;
}
    
```

Expected	Run	
intMax(1, 2, 3) → 3	1	X
intMax(1, 3, 2) → 3	1	X
intMax(3, 2, 1) → 3	3	OK
intMax(9, 3, 3) → 9	9	OK
intMax(3, 9, 3) → 9	3	X
intMax(3, 3, 9) → 9	3	X
intMax(8, 2, 3) → 8	8	OK
intMax(-3, -1, -2) → -1	-3	X
intMax(6, 2, 5) → 6	6	OK
intMax(5, 6, 2) → 6	5	X
intMax(5, 2, 6) → 6	5	X

Your [progress graph](#) for this problem

This Lecture

- 1 Input Arguments
- 2 Writing Proper Code
- 3 Methods
- 4 Kinds of Methods
- 5 Scope
- 6 Methods Examples
- 7 If Statements
- 8 Ifs and Elses
- 9 If Statement Errors
- 10 ANDs and ORs

Section 1

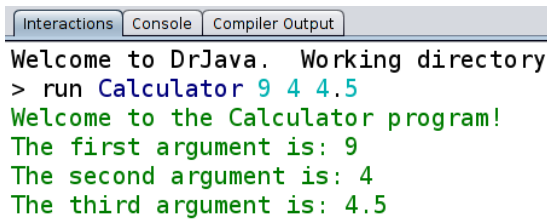
Input Arguments

Assignment Input Arguments

- In the assignment, we input data using input arguments

To use:

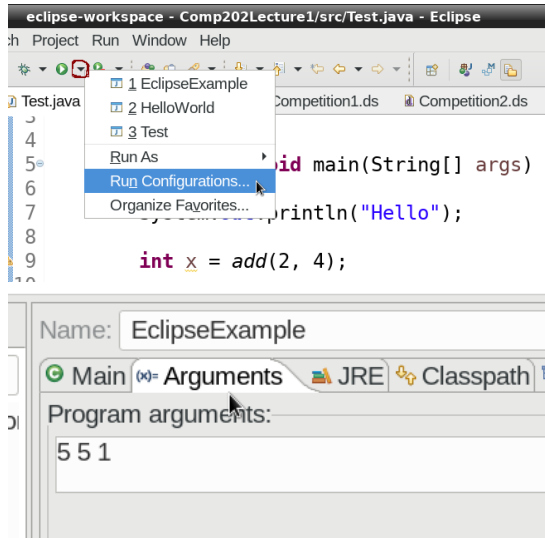
- Open up Calculator.java
- Type *run Calculator 5 5 1* and press Enter



```
Interactions Console Compiler Output
Welcome to DrJava. Working directory
> run Calculator 9 4 4.5
Welcome to the Calculator program!
The first argument is: 9
The second argument is: 4
The third argument is: 4.5
```

Input Arguments in Eclipse

- You access the *Run Configuration* for your program in Eclipse
- Then place the arguments under the *Parameters* tab



- Let's look at the methods to turn Strings into numbers
- `int x = Integer.parseInt("123");`
- `double y = Double.parseDouble("56.4");`
- `boolean z = Boolean.parseBoolean("true");`
- An error is produced if the String doesn't match the type
- `Integer.parseInt("3.5")` doesn't work
- Neither does `Double.parseDouble("Hello")`

Section 2

Writing Proper Code

- The goal of programming is to write correct code
- But it's also extremely important to write readable code
- This section will talk about the difference between good code and bad code
- TAs can take marks off for code that does not meet these guidelines
 - Don't worry, we'll be very forgiving for the first assignment

The golden rule is that your code should be obviously correct and understandable

Multi-step Calculations

Breaking up your calculations and using lots of variables can keep your code readable

```
int value = 5;

//get the remainder mod 2
int remainder = value % 2;

//check if this remainder is 1
//if so, then the number was odd
boolean isOdd = (remainder == 1);

//print out whether the value is odd or not
System.out.println(value + " is odd: " + isOdd);
```

versus

```
System.out.println(value + " is odd: " + (value % 2 == 1));
```

```
int value = 5;

//get the remainder mod 2
int remainder = value % 2;

//check if this remainder is 1
//if so, then the number was odd
boolean isOdd = (remainder == 1);

//print out whether the value is odd or not
System.out.println(value + " is odd: " + isOdd);
```

- Comments are absolutely critical to understanding what your program is trying to do
- You should have a line of comments for every 1-2 lines of code
- Good guideline for if a line of code needs a comment:
 - Is it obvious what this line of code does?
 - The first line is straightforward and we can't write a useful comment here

Indentation

- To keep code organized, programmers **indent** their code
- Between the braces, we tab or space the code in

```
1 public class SayHelloClass
2 {
3     public static void main(String[] args)
4     {
5         sayHello();
6     }
7
8     //This method prints hello world.
9     public static void sayHello()
10    {
11        System.out.println("Hello World");
12    }
13 }
```

- Note how the code within the class and the main method is moved over a bit each time
- Dr. Java can help you indent your code - 'Edit → Indent' command

There are two ways to put your starting braces in your code

On the same line:

```
public static int addNumbers(int a, int b){  
    int c = a + b;  
    return c;  
}
```

On the next line:

```
public static int addNumbers(int a, int b)  
{  
    int c = a + b;  
    return c;  
}
```

Doesn't matter - just be consistent!

Java has a few rules for naming your variables

- Can't start with numbers
- Can only contain letters and numbers
- No keywords (public, static, void, etc.)

And there are some best practices for names:

- Should be short

- Not okay:

- `thisIsAVariableThatStoresAnIntAndWillStoreARemainder`

- Shouldn't be **too** short

- Hard to read code that has a ton of one-letter variable names

- Capitalization should be camelCase

- The first word is lowercase, all others are uppercase

- Example: `isOdd`, `nameOfSchool`

- Should be descriptive and unique

- The better your variables are named, the fewer comments you need

- No random words for variables, such as `kittens`

Bad Naming Example

One student back in my TA days had code that looked like this:

```
int kittens = 145;
int kitties = kittens * 3;
int cats = kitties % 2;
String kats = " is odd: ";
boolean kittyKats = (cats == 1);
System.out.println((kittens + kittens + kittens) + kats + kittyKats);
```

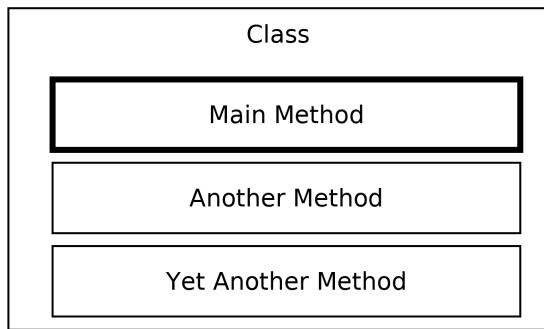
...The TAs will take marks off for this...

Section 3

Methods

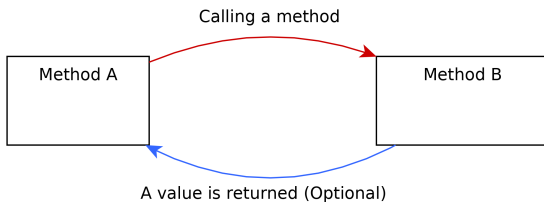
- Methods are pieces of **reusable code**
- They take **parameters** as input
- They **return** a value as output
- They can contain any number of instructions within
- Therefore, a method is just an algorithm

Structure of a Program Notes



Important Points:

- The main method can go above or below the other methods - it doesn't matter
- When the program runs, Java executes instructions starting with the *main method*
- The instructions in other methods are **not executed** unless the method is called



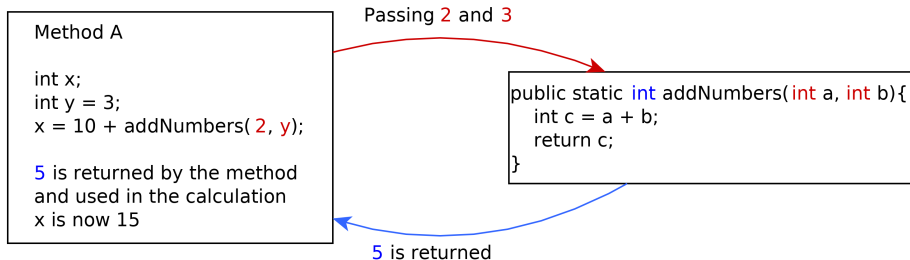
Important Points:

- We can call the method with or without passing parameters
- The method does some calculations
- A value may or may not be returned

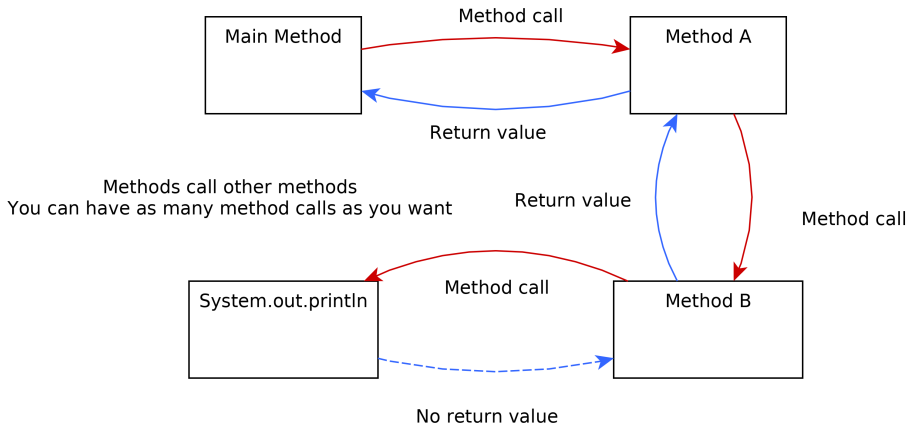
To emphasize some confusion around void:

- `System.out.println` has nothing to do with something being void or not
- Printing something **does not** count as returning a value
- The `return` keyword returns a value
- `void` methods don't have to have `System.out.println`
- `System.out.println` can be in non-void methods

Method Call Example



Chained Method Calls



- Note that `System.out.println` doesn't return a value
 - Return type is **void**
- The method call graph can get complicated
- Methods can call themselves

Section 4

Kinds of Methods

Kinds Of Methods

We can divide methods into four different kinds:

Has Parameters	Has Return Value
X	X
X	✓
✓	X
✓	✓

I'll show example methods and how to call each one

Just the methods are shown below, they'll need to be inside a class to work

No Parameters/No Return Value

- No parameters, and no return value
- May be useful for printing a welcome message
- No return value means this method is void

```
3 //the main method - execution starts here
4 public static void main(String[] args){
5     overTheTopHello();
6 }
7
8 //a method that only prints a message
9 public static void overTheTopHello(){
10     System.out.println("Hello students of COMP 202!");
11     System.out.println("Isn't this a beautiful morning!!");
12     System.out.println("We're all going to have an amazing day!!!!!!");
13 }
```

Parameters/No Return Value

- Has parameters, but no return value
- Could be used to print out a total
- No return value means this method is void

```
//notice that the return type is void
//and the method accepts a double parameter
public static void printTotal(double total)
{
    System.out.println("Your bill is for : $" + total);
}

public static void main(String[] args)
{
    //calling the method with a double variable
    double total = 198.13;
    printTotal(total);

    //could also do
    printTotal(3847.23);
}
```

No Parameters/Return Value

- Has no parameters, but has a return value
- Less commonly seen
- Make sure to put in the return type

```
//notice that the return type is double
//but the method accepts no parameters
public static double getRandomGrade()
{
    //Math.random returns a random double between 0 and 1
    double rand = Math.random();

    //the return value will be between 0 and 100
    return rand * 100;
}

public static void main(String[] args)
{
    //calling the method
    double grade = getRandomGrade();
    System.out.println("Your grade is now: " + grade);
}
```

Parameters/Return Value

- Has parameters and a return value
- This is the most common kind of method

```
//notice the int return type and
//the two int parameters
public static int multiplyNumbers(int a, int b)
{
    int result = a * b;
    return result;
}

public static void main(String[] args)
{
    int x = 3;
    //calling the method
    //passing the parameters
    //storing the result
    int y = multiplyNumbers(x, 45);
    System.out.println("Y is: " + y);
}
```

Section 5

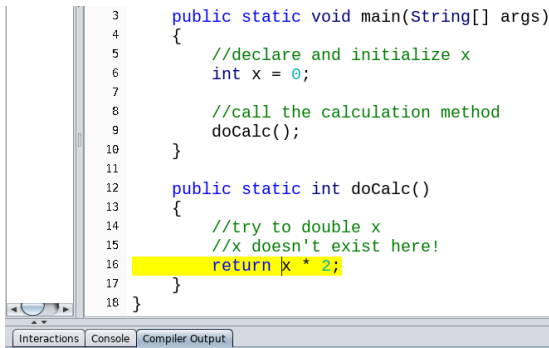
Scope

```
public static int multiplyNumbers(int a, int b)
{
    int result = a * b;
    return result;
}
```

Why is it `int a` and `int b`?

- Need to give the parameters names so you can refer to them in the method's instructions.
- They are being *declared* here
- These variable names have **no connection** to variables in other methods
- `a` and `b` are assigned values when the method is called
- The `a` and `b` variables exist only for that method
- They could be named `banana` and `apple`, or any other name

- Variables only exist in the method they are declared in
 - We call this the variable's **scope**



```
3 public static void main(String[] args)
4 {
5     //declare and initialize x
6     int x = 0;
7
8     //call the calculation method
9     doCalc();
10 }
11
12 public static int doCalc()
13 {
14     //try to double x
15     //x doesn't exist here!
16     return x * 2;
17 }
18 }
```

Interactions Console Compiler Output

1 error found:

File: /home/dcx/Dropbox/COMP 202/Lecture 4 - More Meth

Error: cannot find symbol

symbol: variable x

- We can't access the x variable from another method

This means that there can be two variables named the same thing in two different methods

They will be totally separate and will not share a value

```
public static void main(String[] args){  
    int x = 10;  
    //prints out 10  
    System.out.println("Main x is: " + x);  
}  
  
public static void otherMethod(){  
    int x = 36;  
    //prints out 36  
    //the int x's are not connected in any way!  
    System.out.println("otherMethod x is: " + x);  
}
```

The only way for values to travel from one method to another is:

- By passing the value as a parameter
- By returning a value
- There's a third way, but it'll come later in the course

Remember:

Variables declared in one method cannot be accessed from another method!

Values must be passed as parameters or returned

- Note that variables are not passed
- It's the value that is passed
- This will be important later

Section 6

Methods Examples

Random Number in a Range

- Let's create a helpful method for creating random numbers
- The method will create random numbers between a *min* value and a *max* value

```
public static double getRandNum(int min, int max){  
    //how big is the range between the numbers?  
    int range = max - min;  
  
    //get a random value in this range  
    double rand = Math.random() * range;  
  
    //add it to the min number  
    double finalNum = rand + min;  
  
    //return the random number  
    return finalNum;  
}
```

Rand Range Explanation

```
public static double getRandNum(int min, int max){  
    //how big is the range between the numbers?  
    int range = max - min;  
  
    //get a random value in this range  
    double rand = Math.random() * range;  
  
    //add it to the min number  
    double finalNum = rand + min;  
  
    //return the random number  
    return finalNum;  
}
```

- Example: *min* is 75 and *max* is 100
- We want a number between 75 and 100
- The range is 25, so we get a random number from 0 to 25 (not including 25)
- Then we add that random number to 75 to get the final result

Section 7

If Statements

- We can create a program for ordering pizzas now

This program will have three parts:

- The `pricePerPizza` method will tell us the price per pizza
- The main method will have the number of pizzas to order and will figure out the total cost
- The `printTotal` method will print the number of pizzas and the total price

Pizza Discount

- We have added an **if statement** to the method
- If what's inside the brackets (the **if condition** evaluates to true)
- Then the code inside the **if block** will be executed

```
public static double pricePerPizza(int numPizzas){  
  
    //start the price off at 14.95  
    double price = 14.95;  
  
    //if ten or more pizzas are being ordered  
    if (numPizzas >= 10){  
        //subtract $5 from the price  
        price = price - 5;  
    }  
  
    //return the price per pizza  
    return price;  
}
```

```
public static double pricePerPizza(int numPizzas){  
    //start the price off at 14.95  
    double price = 14.95;  
  
    //if ten or more pizzas are being ordered  
    if (numPizzas >= 10){  
        //subtract $5 from the price  
        price = price - 5;  
    }  
  
    //return the price per pizza  
    return price;  
}
```

- The instruction(s) within the if block only executes if the if condition is true
- So the program can take two different paths
 - *price* could be 14.95 or 9.95 depending on the test

Section 8

Ifs and Elses

Another If Example

- Let's write a method to give a percentage discount based on a person's age
- Let's put this in a method `getDiscountPercentage`
- It will accept `int age` and return a double value

```
public static double getDiscount(int age)
```

Another If Example

```
1 public class AgeDiscount{
2
3     public static void main(String[] args){
4
5         int age = Integer.parseInt(args[0]);
6         System.out.println("You entered: " + age);
7
8         double discount = getDiscount(age);
9
10        System.out.println("The discount is: " + discount);
11    }
12
13    //return the discount for the person's age
14    public static double getDiscount(int age){
15
16        //set the initial value
17        double discount = 0;
18
19        //test the parameter age
20        if (age > 55){
21            discount = 5;
22        }
23        return discount;
24    }
25 }
```

Adding Some Output

```
//return the discount for the person's age
public static double getDiscount(int age){

    //set the initial value
    double discount = 0;

    //test the parameter age
    if (age > 55){
        discount = 5;
        System.out.println("You get a discount!");
    }
    return discount;
}
```

- Now we know if we get a discount
- But what about printing a message if we don't get a discount?
- There are no instructions that execute only when we don't get a discount

We could try this...

```
13      //return the discount for the person's age
14      public static double getDiscount(int age){
15
16          //set the initial value
17          double discount = 0;
18
19          //test the parameter age
20          if (age > 55){
21              discount = 5;
22              System.out.println("You get a discount!");
23          }
24          if (age < 55){
25              System.out.println("You don't get a discount!");
26          }
27          return discount;
28      }
```

- Note that there's an error here
- Neither block will execute if the age is exactly 55

The Else Statement

We use the *else statement*

```
13 //return the discount for the person's age
14 public static double getDiscount(int age){
15
16     //set the initial value
17     double discount = 0;
18
19     //test the parameter age
20     if (age > 55){
21         discount = 5;
22         System.out.println("You get a discount!");
23     }
24     else{ //this block executes when the first block does not
25         System.out.println("You don't get a discount!");
26     }
27     return discount;
28 }
```

- Either the *if* or the *else* block will execute

- What if we have multiple cases?
- Over 65? A 10% discount
- Over 55? A 5% discount

```
//return the discount for the person's age
public static double getDiscount(int age){

    double discount = 0;

    if (age > 65){
        discount = 10;
    }
    else if (age > 55){
        discount = 5;
    }
    else{ //this block executes when the tests for the others fail
        System.out.println("You don't get a discount!");
    }
    return discount;
}
```

- You can have as many cases as you want
- The tests evaluate from top-to-bottom
 - Java chooses the first one that is true
- The *else* block executes if the other tests fail
- You don't need an *else* block

- Let's build a mini-calculator that takes two numbers and a operation
- For example:
- `run Calc 1 + 2` should print out 3

```
//get the two ints and the operator
//from the input arguments
int a = Integer.parseInt(args[0]);
String operator = args[1];
int b = Integer.parseInt(args[2]);
```

Get the arguments from the user

Don't worry about the square brackets in `args[0]`, we'll talk about that later

```
double result = 0;  
  
if (operator.equals("+")){  
    result = a + b;  
}  
else if (operator.equals("*")){  
    result = a * b;  
}  
//...Add in other operators here
```

Do the calculation based on the operator

```
else {  
    //Show error for unknown operators  
    System.out.println("Error: " + operator);  
}  
  
System.out.println("Result: " + result);
```

Have a last else to handle any other operators
And print out the result

```
//get the two ints and the operator
//from the input arguments
int a = Integer.parseInt(args[0]);
String operator = args[1];
int b = Integer.parseInt(args[2]);

double result = 0;

if (operator.equals("+")){
    result = a + b;
}
else if (operator.equals("*")){
    result = a * b;
}
//...Add in other operators here

else {
    //Show error for unknown operators
    System.out.println("Error: " + operator);
}

System.out.println("Result: " + result);
```

Another If Example

- Let's look at creating an absolute value method
- **Input:** A number which can be negative or positive
- **Output:** The positive version of that number
- Example: `abs(4)` returns 4
- Example: `abs(-9)` returns 9
- **Instructions:** If the number is negative, multiply it by -1. Then return the number


```
//return the absolute value of a number
//if the number is negative,
//multiply it by negative one to
//make it positive
public static int abs(int x)
{
    if (x < 0)
    {
        x = x * -1;
    }
    return x;
}
```

- If the value is below zero, multiply it by -1
- Then return x

```
//another version of the method
public static int abs2(int x){
    if (x < 0){
        return x * -1;
    }else{
        return x;
    }
}
```

- Another equivalent version
- If x is below zero, return x times negative one
- Else, just return x

```
//yet another version of the method
public static int abs3(int x){
    if (x < 0){
        return x * -1;
    }

    return x;
}
```

- Yet another equivalent version
- If x is below zero, return x times negative one
- Lastly, return x
- We don't need the else here
- The last instruction is only executed if the if condition was false

Section 9

If Statement Errors

What's wrong with this *if block*?

```
public static int abs(int x)
{
    if (x < 0);
    {
        x = x * -1;
    }
    return x;
}
```

There's an extra semicolon at the end of the if statement

```
public static int abs(int x)
{
    if (x < 0);
    {
        x = x * -1;
    }
    return x;
}
```

- The block of code within the if will **always** execute. Very confusing!
- Because the *if statement* starts a **block of code**, we don't add a semi-colon
- To put it another way, either we add braces after a line of code, or a semi-colon, never both
- For example: Methods have braces, so we don't add braces

What's wrong with this code?

```
13      //return the discount for the person's age
14      public static double getDiscount(int age){
15          if (age > 65){
16              double discount = 10;
17          }
18          else if (age > 55){
19              double discount = 5;
20          }
21          return discount;
22      }
23  }
```

Interactions Console Compiler Output

[line: 21]
Error: cannot find symbol
symbol: variable discount

Why does it say the variable cannot be found?

- Remember how we talked about **scope**, where variables only existed in the method they were created in?
- Actually, variables only exist in the block (between the braces), where they are created.

Here's the fixed version:

```
//return the discount for the person's age
public static double getDiscount(int age){

    double discount = 0; //default value
    if (age > 65){
        discount = 10;
    } else if (age > 55){
        discount = 5;
    }
    return discount;
}
```

- Now there is a default value that will be returned from this method


```
//return the discount for the person's age
public static double getDiscount(int age){

    if (age > 65){
        return 10;
    }else if (age > 55){
        return 5;
    }else{
        return 0;
    }
}
```

- Another version, where each path returns a value

Scope Example

```
public static void main(String[] args)
{
    //x exists in all of main
    int x = 0;

    if (x >= 0){
        //x exists within the block
        x = 5;

        int y = x + 1;
        //y only exists within here
    }

    //y doesn't exist out here
    //y = x + 1;

    //z exists in main after this point
    double z = 0;
}
```

- Variables exist after they are declared
- Variables continue to exist if new blocks are opened
- They stop existing after the block they were created in is closed
- For example, y stops existing when the if block is finished

Scope Example

```
1 public class ScopeExample
2 {
3     public static void main(String[] args){
4         //x exists in all of main
5         x int x = 8;
6
7         if (x >= 0){
8             y x = 5; //x exists within the block
9             [ int y = x + 1; //y only exists within here
10
11             System.out.println("X is: " + x);
12             System.out.println("Y is: " + y);
13         }
14
15         //y doesn't exist out here
16         //y = x + 1;
17         //System.out.println("Y is: " + y);
18
19         //z exists in main after this point
20         z [ double z = 56;
21             System.out.println("X is: " + x);
22             System.out.println("Z is: " + z);
23         }
24     }
```

```
//return the discount for the person's age
public static double getDiscount(int age){

    double discount = 0;

    if (age > 65){
        discount = 10;
    }
    else if (age > 55){
        discount = 5;
    }
    return discount;
}
```

We saw this before. What happens if we reverse the tests for the ages?

Wrong Ordering

```
//return the discount for the person's age
public static double getDiscount(int age){

    double discount = 0;

    if (age > 55){
        discount = 5;
    }
    else if (age > 65){
        discount = 10;
    }
    return discount;
}
```

- The first case covers the second case
- Everyone who is over 65 is also over 55
- So the second case will never execute

Java evaluates the cases from top to bottom

Section 10

ANDs and ORs

- What if we want to make more complicated decisions?
- For example, a program to decide whether to run outside
- We'll have `temperature` and `isRaining`
- Method: We won't run when it's raining, when it's too cold, or when it's snowing outside

```
//return a boolean whether or not we should run
public static boolean shouldRun(int temp, boolean isRaining)
{
    //assume we can run
    boolean run = true;

    //check if it's too cold
    if (temp < -20){
        run = false;
    }

    //check if it's raining
    if (isRaining){
        run = false;
    }

    //return the run variable
    return run;
}
```

- We assume we can run
- If we have bad conditions, then we can't run

Snowing Expression

Let's figure out how to make an expression to tell if it's snowing (below zero and is snowing)

Note we used the word *and*...

We will connect Booleans expressions together using the *AND* operator

Let's see a messy way of connecting Boolean expressions

```
boolean isSnowing = false;
if (isRaining){           //check if it's raining
    if (temp < 0){         //check if it's freezing
        isSnowing = true;
    }
}
```

This works but is confusing

Let's see a better way:

```
//snowing when it's raining and freezing  
boolean isSnowing = isRaining && temp < 0;
```

- isSnowing is true if (and only if) the boolean values on either side are true
- That is, isRaining must be true and temp must be below zero

```
//snowing when it's raining and freezing  
boolean isSnowing = isRaining && temp < 0;
```

```
//or
```

```
//snowing when it's freezing and raining  
boolean isSnowing = temp < 0 && isRaining;
```

We can reverse the tests if we want.

- Symbol: && - The ampersand
- The result is true only if both of the boolean expressions on the sides are true
- For example: true && false is false

Represented in a *truth table*:

Side A	Side B	Result
false	false	false
false	true	false
true	false	false
true	true	true

```
boolean canSeeMoon = isNight && isClearSky
```

Another common logic operator we use is the OR operator.

- For example, we could say a grade is invalid if the grade is under 0 or above 100

The OR Operator

Let's print a message if the grade entered is too high or too low:

```
1 public class GradeExample{
2
3     public static void main(String[] args){
4         //get the grade from the user
5         int grade = Integer.parseInt(args[0]);
6         System.out.println("You entered: " + grade);
7         |
8         //complain if grade is negative
9         if (grade < 0){
10             System.out.println("That grade is invalid.");
11             System.out.println("Please check the grade and enter it again.");
12         }
13
14         //complain if grade is too high
15         if (grade > 100){
16             System.out.println("That grade is invalid.");
17             System.out.println("Please check the grade and enter it again.");
18         }
19     }
20 }
```

This works, but we're duplicating code which is poor practice

The OR Operator

We can simplify this using the OR operator:

```
1 public class GradeExample{
2
3     public static void main(String[] args){
4         //get the grade from the user
5         int grade = Integer.parseInt(args[0]);
6         System.out.println("You entered: " + grade);
7
8         //complain if grade is negative or is too high
9         if (grade < 0 || grade > 100){
10             System.out.println("That grade is invalid.");
11             System.out.println("Please check the grade and enter it again.");
12         }
13     }
14 }
```


- Symbol: `||` - Vertical bar - Found above the enter key
- The result is true if either one of the boolean expressions on the sides is true
- For example: `true || false` is true

Represented in a *truth table*:

Side A	Side B	Result
false	false	false
false	true	true
true	false	true
true	true	true

```
boolean isDark = isNight || isLightsOut
```

One Last Operator

What if we want to test the opposite of something?

```
//snowing when it's raining and freezing
boolean isSnowing = isRaining && temp < 0;

if (isSnowing){
    //do nothing
}
else
{
    System.out.println("Booo! I like snow :(");
}
```

We wrote useless code here, which is a bad practice
How can we easily check to see if it is NOT snowing?

The NOT Operator

```
//snowing when it's raining and freezing
boolean isSnowing = isRaining && temp < 0;

if (!isSnowing){
    //when isSnowing is false, this instruction will execute
    System.out.println("Booo! I like snow :(");
}
```

The exclamation mark means NOT
Say the *if statement* as *if not isSnowing*

NOT

- Symbol: !
- Gives the opposite boolean value
- For example: !true is false

Represented in a truth table:

Side A	Result
false	true
true	false

```
boolean isCat = isFuzzy && !isADog
```

- As in life, keep it positive!
- That is, when you're naming boolean variables, use something like `isValid`
- Then it's easy to reason about `!isValid`
 - The *if block* executes when something is not valid
 - `if (!isValid)` is very confusing

- Try to keep your expressions simple
- `boolean isSnowing = isRaining && temp < 0` is pretty simple
- `boolean isSnowing = !(isRaining || temp >= 0)` is logically the same, but more confusing

As practice for your boolean expressions, there are a few examples on the assignment warm-up

For example, is the following true or false:

$(\text{true and false}) \text{ or } !(\text{true or true}) \text{ or } !(\text{false and } (\text{false or true}))$

It's true

Go through it one step at a time and simplify

Reminder: `&&` is *and*, `||` is *or*, and `!` is *not*

```
(true && false) || !(true || true) || !(false && (false || true))  
      (false) || !(true) || !(false && (true))  
            false || false || !(false)  
            false || false || true  
            true
```


What if we have variables?

If `a = true` and `b = false`:

Reminder: `&&` is *and*, `||` is *or*, and `!` is *not*

$$(a \ \&\& \ b) \ || \ (! (a \ || \ b) \ \&\& \ ! (b \ \&\& \ b))$$

Substitute in *a* and *b*

$$(true \ \&\& \ false) \ || \ (! (true \ || \ false) \ \&\& \ ! (false \ \&\& \ false))$$
$$(false) \ || \ (! (true) \ \&\& \ ! (false))$$
$$false \ || \ (false \ \&\& \ true)$$
$$false \ || \ false$$

false