

Lecture Jan 29 - Finishing Logic and Casting

Bentley James Oakes

January 28, 2018

- EPTS (Engineering Peer Tutoring Services) Comp Tutor Office hours
- Wednesday 13:05 - 14:25
- Friday 9:05 - 10:25
- Frank Dawson Adams (FDA) 6

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

Warmup-1 > hasTeen

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

We'll say that a number is "teen" if it is in the range 13..19 inclusive. Given 3 int values, return true if 1 or more of them are teen.

hasTeen(13, 20, 10) → true

hasTeen(20, 19, 10) → true

hasTeen(20, 10, 13) → true

Go

...Save, Compile, Run

Show Solution

```
public boolean hasTeen(int a, int b, int c) {  
    |  
}  
|
```

- 1 If Structures
- 2 If Statement Errors
- 3 ANDs
- 4 ORs
- 5 NOTs
- 6 Boolean Resolving
- 7 Casting

Section 1

If Structures

If Structure Rules

- The first test must be an `if`
- You can have as many `else ifs` as you want, including zero
- You can include an `else` at the end, but you don't have to have one
- Java will test each case from the top to the bottom
- When a condition is `true`, that block will be executed
- After a block executes, Java skips down past the other cases
- An `else` block executes when no other case has been `true`

- 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]);

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 2

If Statement Errors

What's wrong with this *if block*?

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

Extra Semi-colon

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 a semi-colon

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 //x exists in all of main
5         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 }
```

Section 3

ANDs

- 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

Using AND

Let's see a messy way of connecting Boolean expressions

```
boolean isSnowing = false;

if (isRaining){
    if (temp < 0){
        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
```

AND Example

- Let's try another example with AND
- A method `isInMiddle`
- Input: Three doubles `left`, `middle`, and `right`
- Output: A boolean: *true* if `middle` is between `left` and `right`, and *false* otherwise

AND Example

```
//check if the middle variable is between the left and the right
public static boolean isInMiddle(int left, int middle, int right)
{
    boolean inMiddle = middle >= left && middle <= right;
    return inMiddle;
}
```

- Don't write this:

```
boolean bad = left <= middle <= right;
```

AND Example

```
//an alternative implementation
public static boolean isInMiddle2(int left, int middle, int right)
{
    //handle the failure on the left
    if (middle < left){
        return false;
    }
    //handle the failure on the right
    else if (middle > right){
        return false;
    }
    //middle must be in the middle
    else{
        return true;
    }
}
```

- Note that there are multiple points where we can return from the method
- All cases must be covered by a return
 - That is, there must be a return statement on every path through the method

Section 4

ORs

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:

```
public static void main(String[] args)
{
    //get the grade from the user
    int grade = Integer.parseInt(args[0]);
    System.out.println("You entered: " + grade);

    boolean isValid = false;

    //set boolean to true if grade is negative
    if (grade < 0){
        isValid = true;
    }

    //set boolean to true if grade is too high
    if (grade > 100){
        isValid = true;
    }

    if (isValid){
        System.out.println("That grade is invalid.");
    }
}
```

The OR Operator

We can simplify this using the OR operator:

```
public static void main(String[] args)
{
    //get the grade from the user
    int grade = Integer.parseInt(args[0]);
    System.out.println("You entered: " + grade);

    boolean isValid = grade < 0 || grade > 100;

    //print out the error message
    if (isValid){
        System.out.println("That grade is invalid.");
    }
}
```

- 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
```

Example CodingBat Question

Let's see a CodingBat question:

<http://codingbat.com/prob/p178986>

Warmup-1 > hasTeen

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

We'll say that a number is "teen" if it is in the range 13..19 inclusive. Given 3 int values, return true if 1 or more of them are teen.

hasTeen(13, 20, 10) → true

hasTeen(20, 19, 10) → true

hasTeen(20, 10, 13) → true

Go

...Save, Compile, Run

Show Solution

```
public boolean hasTeen(int a, int b, int c) {  
    |  
}  
|
```

Here's my solution:

Warmup-1 > hasTeen

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

We'll say that a number is "teen" if it is in the range 13..19 inclusive. Given 3 int values, return true if 1 or more of them are teen.

hasTeen(13, 20, 10) → true

hasTeen(20, 19, 10) → true

hasTeen(20, 10, 13) → true

Go

...Save, Compile, Run

Show Solution

```
public boolean hasTeen(int a, int b, int c) {  
    //return true if any of them are true  
    //first is true OR second is true OR third is true  
    return isTeen(a) || isTeen(b) || isTeen(c);  
}  
  
public boolean isTeen(int x) {  
    //return true if x is in this range  
    return x >= 13 && x <= 19;  
}
```

Here's their solution:

```
public boolean hasTeen(int a, int b, int c) {  
    // Here it is written as one big expression  
    // vs. a series of if-statements.  
    return (a>=13 && a<=19) ||  
           (b>=13 && b<=19) ||  
           (c>=13 && c<=19);  
}
```

Section 5

NOTs

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

Section 6

Boolean Resolving

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

For example, is the following true or false:

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

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 $a = \text{true}$ and $b = \text{false}$:

Reminder: $\&\&$ is *and*, $\|\|$ is *or*, and $!$ is *not*

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

Substitute in a and b

$$(\text{true} \&\& \text{false}) \|\| (! (\text{true} \|\| \text{false}) \&\& !(\text{false} \&\& \text{false}))$$
$$(\text{false}) \|\| (! (\text{true}) \&\& !(\text{false}))$$
$$\text{false} \|\| (\text{false} \&\& \text{true})$$
$$\text{false} \|\| \text{false}$$
$$\text{false}$$

Section 7

Casting

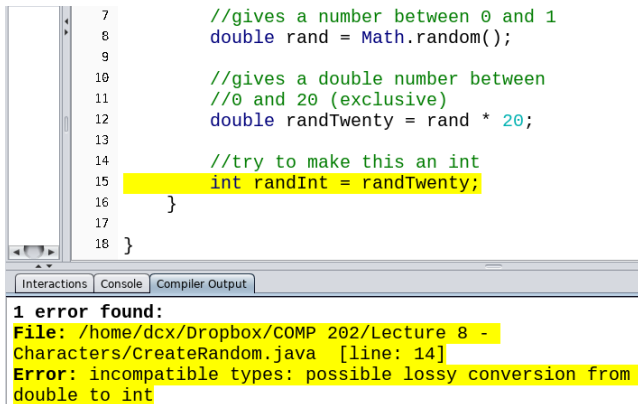
- In Java, the type of a variable matters
- Sometimes we have a value in one variable that we want to convert to another type
- We did see how to get a random number from 0 (inclusive) to 1 (exclusive)
- And how to scale that number up
- But this gives a double value
- What if we want a random integer value?

```
//gives a number between 0 and 1  
double rand = Math.random();
```

```
//gives a double number between  
//0 and 20 (exclusive)  
double randTwenty = rand * 20;
```

- The number produced is a double value

- What happens if we place this in an integer?
- Java will give us an error because integers store less data



```
7 //gives a number between 0 and 1
8 double rand = Math.random();
9
10 //gives a double number between
11 //0 and 20 (exclusive)
12 double randTwenty = rand * 20;
13
14 //try to make this an int
15 int randInt = randTwenty;
16 }
17
18 }
```

1 error found:
File: /home/dcx/Dropbox/COMP 202/Lecture 8 - Characters/CreateRandom.java [line: 14]
Error: incompatible types: possible lossy conversion from double to int

- We want to force this value into an integer

```
//cast this double to an int  
int randInt = (int) randTwenty;
```

- Note how we add in the (int) before the value
- This is called **casting**
- We are **casting** the double value to the integer type

```
double x = 20.456;  
System.out.println("X is: " + x);
```

```
int y = (int) x;  
System.out.println("Y is: " + y);
```

X is: 20.456

Y is: 20

- When we cast to an `int`, we throw away the decimal

Careful About Casting

- Be careful of how casting works

```
int num = (int) Math.random();
```

- This will always give zero. Why?
- Because the number will always be 0.0 to 0.999...
- If we throw away the decimal part, we **always get zero**

- Avoid this error

```
int num = (int) Math.random() * 20;
```

- The cast to an int is very 'sticky', and it happens before other operators
- It be be applied to the `Math.random()` **before** the multiplication
- So we are always multiplying by zero

- Wrap the scaling calculation in brackets

```
int num = (int) (Math.random() * 20);
```

- What values can this give?
- 0 to 19 as an integer

Random Ranges

- What if we don't want to start at zero?
- If you want to generate a number from 5 to 20
- Think about adding 0 to 15 to 5

```
int num = 5 + (int) (Math.random() * 16);
```

- What values can this give?
- 5 to 20
- Explanation: $5 + 0$ to 15

- Great practice (and a warmup question on the assignment)
- Is to create a method with two parameters
 - A minimum value and a maximum value
- And the method returns a random int between those two numbers

The following slides have conversion examples between:

- Integers
- Doubles
- Strings
- Booleans

Sounds more complicated than it is

We'll also see characters later

Let's see how to convert integers to other types:

Type	Example	Notes
double	<code>double d = 5;</code>	Happens automatically
String	<code>String s = 15 + "";</code>	Uses concatenation
boolean	-	Not possible

Let's see how to convert doubles to other types:

Type	Example	Notes
integer	<code>int x = (int) 6.6;</code>	Loss of information
String	<code>String s = 7.8 + "";</code>	Uses concatenation
boolean	-	Not possible

String Conversions

Let's see how to convert String to other types:

Type	Example	Notes
integer	<code>String s = Integer.parseInt("456");</code>	-
double	<code>String t = Double.parseDouble("134.5");</code>	-
boolean	<code>String p = Boolean.parseBoolean("true");</code>	-

Let's see how to convert booleans to other types:

Type	Example	Notes
integer	-	Not possible
double	-	Not possible
String	String s = true + "";	-

- The following slides are just for information
- We won't test you on them
- But it's to get a sense of the different variable types in Java

Primitive Type Details

These variables types hold numbers without decimals

Type	Size	Range
byte	8 bits	-128 to 127
short	16 bits	-32768 to 32767
int	32 bits	Around ± 2.1 billion
long	64 bits	Around $\pm 9.2 \times 10^{18}$

Type	Size	Range
boolean	1 bit	true, false
char	16 bits	Stores a char '0', 'A', '?', etc.

- Fun fact: apparently chars don't handle emojis well
- <https://codeahoy.com/2016/05/08/the-char-type-in-java-is-broken/>

Type	Size	Closest to 0 Val	Farthest from 0 Val
float	32 bits	$\pm 1.4 \times 10^{-45}$	$\pm 3.8 \times 10^{38}$
double	64 bits	$\pm 4.9 \times 10^{-324}$	$\pm 1.8 \times 10^{308}$

- Also have special values: 0, $+\infty$, $-\infty$, NaN (Not a Number)
- See <http://introcs.cs.princeton.edu/java/91float/>

- Don't worry about the details for COMP 202
- But in a real program:
 - Need to pick an appropriate variable type
 - And worry about losing information with casts

Casting Problems

```
//create a four billion value
double bigNumber = 4000000000.0;
System.out.println("Big #: " + bigNumber);
//prints Big #: 4.0E9

//cast the number down to an int
int num = (int) bigNumber;
System.out.println("As int #: " + num);
//prints 2147483647

System.out.println("Max Value: " + Integer.MAX_VALUE);
//max value for an int: 2147483647
```

Precision Problems

- Don't use float/double for currency
- There are issues with binary representation

```
double cash = 0.1 + 0.1 + 0.1;  
System.out.println("Value: " + cash);  
//Value: 0.30000000000000004
```

```
System.out.println("Is equal: " + (cash == 0.3));  
//Is equal: false
```