

# Lecture Jan 31 - Casting and Loops

Bentley James Oakes

January 30, 2018

- Due **TODAY** before midnight
- Office Hours: 13:30 - 15:00 in Room 233 McConnell
- Assignments handed in tomorrow receive -10% penalty
- Assignments handed in Friday receive -20% penalty
- Please hand an assignment in!

# Assignment 2

- Posted already
- Due **Thursday, February 15th**

- 1 Recap
- 2 Casting
- 3 Conversions
- 4 Loops
- 5 Example: Finding Prime Numbers
- 6 Example: Collatz Conjecture

# Section 1

## Recap

- An example with AND
- A method `isInMiddle`
- Input: Three doubles `num`, `left`, and `right`
- Output: A boolean: *true* if `num` is between `left` and `right`, and *false* otherwise

# AND Example

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

- Note the following is not allowed:

```
boolean badCheck = left <= num <= right;
```

# AND Example

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

- Note that there are multiple points where we can return from the method
- There must be a return statement on every path through the method



# The OR Operator

Let's have a method that checks if an entered grade 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.");
    }
}
```

- To make this even shorter, place the boolean expression right in the *if condition*

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 is x is in this range  
    return x >= 13 && x <= 19;  
}
```

# The NOT Operator

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

if (!isSnowing){
    System.out.println("Yay! No snow today! :)");
}
```

Say this as *If not isSnowing*

# 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     }
```

## Section 2

# 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?

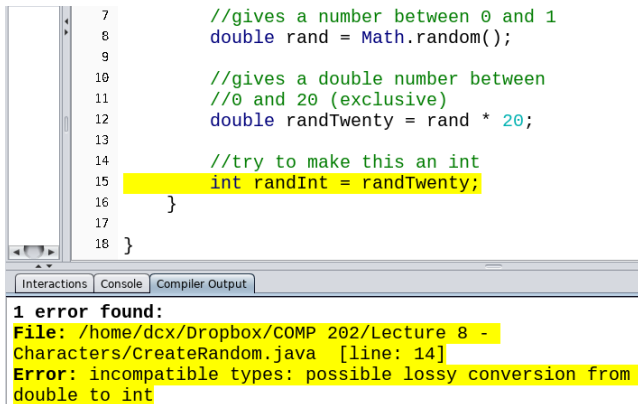


# Random Number Example

```
//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



The screenshot shows an IDE window with a Java file named `CreateRandom.java`. The code defines two `double` variables, `rand` and `randTwenty`, and then attempts to assign `randTwenty` to an `int` variable `randInt`. The IDE's `Compiler Output` pane at the bottom displays a single error: `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`. The error message is highlighted in yellow.

```
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

## Section 3

# Conversions

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>int x = Integer.parseInt("456");</code>	-
double	<code>double y = Double.parseDouble("134.5");</code>	-
boolean	<code>boolean z = 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 $\pm 2.1$ 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

```
//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
```

## Section 4

# Loops

- Often, we want to repeat instructions in recipes/algorithms
- For example, we add three eggs to a bowl
  - Pick up the egg, crack it, dump the yolk, put the shell down
- Or, writing a method to find all the prime numbers from 0 to 100



```
//execute while this boolean expression is true
//it's always true, so loop forever
while(true)
{
    System.out.println("Hello!");
}
```

- A **while statement** repeatedly executes the instructions in the **while block** for as long as the **while condition** is true
- This **while condition** is always true
- Hit *Reset* In Dr. Java to stop the program

# Execute Once

```
//set the variable to true
boolean doOnce = true;

//doOnce is tested twice
//the while block executes once
while(doOnce)
{
    System.out.println("doOnce is: " + doOnce);
    doOnce = false;
    System.out.println("doOnce is then: " + doOnce);
}
System.out.println("Finished with loop!");
```

doOnce is: true  
doOnce is then: false  
Finished with loop!

- The *while condition* is tested, and the variable is true
  - So the block executes
  - The variable's value is printed, changed to false, then printed again
- The *while condition* is tested again, but the variable is false
  - So Java exits the loop and moves on

# Counting

- We can use *while statements* for counting

```
//set i
int i = 0;

//keep looping while i is less than 2
while (i < 2){
    //print out the value of i
    System.out.println("i is now: " + i);

    //increase the value of i
    i = i + 1;
}
```

- What's the first value of *i* printed? 0
- What's the last value of *i* printed? 1
- Why? Because when *i* == 2, the loop exits
- So this loop executes 2 times

```
//set i
int i = 0;

//keep looping while i is less than 2
while (i < 2){
    //print out the value of i
    System.out.println("i is now: " + i);

    //increase the value of i
    i = i + 1;
}
```

- While condition:  
 $0 < 2?$
- Yes, so print and increase
- While condition:  
 $1 < 2?$
- Yes, so print and increase
- While condition:  
 $2 < 2?$
- No, so stop the loop
- i is 2 after the loop

```
//start j at 0
int j = 0;

while (j < 100){
    //perform a test
    if (j % 2 == 0){
        System.out.println("J: " + j);
    }

    j++; //shortcut for j = j + 1
}
```

- What does this code do?
- Prints all the even numbers from 0 to 100
- Notice the shortcut, instead of `j = j + 1`
- we can write `j++` or `j += 1`

# Random Numbers

- Let's do a program that loops until the random number generator gives 0

```
public static void main(String[] args)
{
    double randNum = Math.random();

    while (randNum > 0.1)
    {
        System.out.println(randNum);
        randNum = Math.random();
    }
    System.out.println("Done");
}
```

# The Components of Iteration

```
int k=0;
while (k<5){
    System.out.println("K: " + k);
    k++;
}
```

- There's a pattern to the way we are **iterating** here

# The Components of Iteration

```
int k=0; //creation
while (k<5){ //condition/test
    System.out.println("K: " + k);
    k++; //modification
}
```

- Creation - We create an **iteration variable** and assign it a starting value
- Condition/test - We define the test for how many times the loop should run
- Modification - We change the iteration variable



# The For Loop

- We iterate so much in programming, we have a special loop for it

```
//the three parts of a for loop:  
//initialization - int k=0  
//condition/test - k < 5  
//iteration      - k++  
for (int k=0 ; k<5 ; k++)  
{  
    System.out.println("K: " + k);  
}
```

- It's the three components of **iteration**
- We just put them on one line
- And have two semi-colons in there

# For Loop

```
for (int s=0; s<3; s++)  
{  
    System.out.println("s: " + s);  
}
```

```
s: 0  
s: 1  
s: 2
```

- Creation of s with value 0
- Condition:  $0 < 3$ ?
  - Yes, so print
  - Do the modification:  $s++$  (which means  $s + 1$ ) (so  $s = 1$ )
- Condition:  $1 < 3$ ?
  - Yes, so print
  - Do the modification:  $s++$  (so  $s = 2$ )
- Condition:  $2 < 3$ ?
  - Yes, so print
  - Do the modification:  $s++$  (so  $s = 3$ )
- Condition:  $3 < 3$ ?
  - No, so stop the loop

Let's make sure we know how many times the loop executes

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

```
//creation: start i at 0
//condition: execute while i is less than 15
//modification: add one to i each time (increment)
for (int i=0; i < 15; i++){
    System.out.print(i + " ");
}
System.out.println();
```

Note that the condition is  $i < 15$ , so the last value printed is 14

# Countdown

We can change the three components to count down

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

```
//creation: start i at 15
//condition: execute while i is at or above 0
//modification: subtract one from i each time (decrement)
for (int i=15; i >= 0; i--){
    System.out.print(i + " ");
}
System.out.println();
```

Note that the condition is `i >= 0`, so the last value printed is 0

# Blastoff

Let's change our code to print *Blastoff* at zero

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 Blastoff!

```
//creation: start i at 15
//condition: execute while i is at or above 0
//modification: subtract one from i each time (decrement)
for (int i=15; i >= 0; i--){

    //within the for loop,
    //print something different if i is 0 or not
    if (i == 0){
        System.out.print("Blastoff!");
    }else{
        System.out.print(i + " ");
    }
}
System.out.println();
```

- Let's print all the powers of two below 2000

1 2 4 8 16 32 64 128 256 512 1024

```
//creation: i = 1
```

```
//condition: i < 2000
```

```
//modification: double i each iteration
```

```
for (int i=1; i < 2000; i = i*2)
```

```
{
```

```
    System.out.print(i + " ");
```

```
}
```

```
System.out.println();
```

# Summing Numbers

- Let's get the sum of all the numbers from 0 to 100

```
int sum = 0;
for (int r=0; r <= 100; r++)
{
    sum = sum + r;
}
System.out.println("The sum from 0 to 100: " + sum);
```

- Be careful of the condition: here we want `r` to be equal to 100 on the last iteration
- Note that we can only use the iteration variable inside the loop
- Recall the scope discussion: `r` doesn't exist outside the braces

## Section 5

### Example: For Loop Drawing



- We can start to draw some art with *for loops*
- Let's start with drawing lines and boxes using stars and spaces

- Let's draw a line with ten stars without a *for loop*

```
System.out.println("*****");
```

```
> run Drawing
```

```
*****
```

# Drawing a Line

- Let's draw a line with ten stars with a *for loop*

```
for (int i=0; i < 10; i++)  
{  
    System.out.print("*");  
}  
  
System.out.println();
```

- Note that `System.out.print` is used
- Also note that the condition tests for `i < 10`
  - First iteration:  $i = 0$
  - Last iteration:  $i = 9$
  - Ten iterations in total

# Drawing a Line

- Let's draw a line with any number of stars with a *for loop*

```
int numStars = 20;

for (int i=0; i < numStars; i++)
{
    System.out.print("*");
}

System.out.println();
```

- We created a new variable for the number of stars to print
- We added this variable to the condition in the *for loop*

# Drawing a Line

- Let's draw a line with a random number of stars with a *for loop*

```
//gives a number from 0 to 1
double randNum = Math.random();

//store a number from 0 to 20
//note: because randNum is a double
//numStarsDbl will be a double
double numStarsDbl = randNum * 20;

for (int i=0; i < numStarsDbl; i++)
{
    System.out.print("*");
}

System.out.println();
```

- Note that we can use a double in our condition

# Using Conditions When Drawing

- Let's try drawing a line that alternates hyphens and spaces
- We'll start with drawing a line

- - - - -

```
int lineLength = 20;
for (int i=0; i < lineLength; i++){
    System.out.print("- ");
}
System.out.println();
```

# Drawing with Spaces

- Let's add a condition for deciding whether to print a hyphen or a space
- We will base this on whether  $i$  is odd or not

```
- - - - -  
  
int lineLength = 20;  
for (int i=0; i < lineLength; i++){  
    if (i%2 == 0){  
        System.out.print("-");  
    }else{  
        System.out.print(" ");  
    }  
}  
System.out.println();
```

# Drawing a Line with Arrows

- Let's try drawing a line that ends in arrows
- The left end will have <, the right end will have >, and the line will be made up of -
- We'll start with drawing a line

```
int lineLength = 20;  
for (int i=0; i < lineLength; i++){  
    System.out.print("-");  
}  
System.out.println();
```



# Drawing a Line with Arrows

- Now that we have a line, let's add the arrow at the beginning

> run Drawing

<-----

```
int lineLength = 20;
for (int i=0; i < lineLength; i++){
    if (i == 0){
        System.out.print("<");
    }else{
        System.out.print("-");
    }
}
System.out.println();
```

# Drawing a Line with Arrows

- Now we add the arrow at the end

<----->

```
int lineLength = 20;
for (int i=0; i < lineLength; i++){
    if (i == 0){
        System.out.print("<");
    }else if (i == lineLength - 1){
        //this is true when i == 19
        //aka the last iteration
        System.out.println(">");
    }else{
        System.out.print("-");
    }
}
System.out.println();
```

# Drawing Methods

- Let's draw an empty box

```
 * * * * *  
 *       *  
 *       *  
 *       *  
 *       *  
 * * * * *
```

This will have three parts:

- A line at the top
- Multiple lines of star - spaces - star
- Another line at the bottom

- Let's use a method to draw the top and bottom line

```
public static void drawLine(int size)
{
    //draw a line of stars
    //'size' number of stars
    for (int i=0; i < size; i++)
    {
        System.out.print("*");
    }
    System.out.println("");
}
```

Now we just call this method to print a line

- Now a method to create the edges
- This will produce one star, some spaces, and then another star

```
public static void drawEdges(int size)
{
    //iterate 'size' times
    for (int i=0; i < size; i++)
    {
        //if it is the first or last
        //iteration, print a star
        if (i == 0 || i == size-1){
            System.out.print("*");

            //otherwise, print a space
        }else{
            System.out.print(" ");
        }
    }
    System.out.println("");
}
```

- Now the main method can tie these methods together

```
public static void main(String[] args)
{
    //set the box size
    int boxSize = 5;

    //draw the top line
    drawLine(boxSize);

    //draw the middle of the box
    //note that we draw boxsize - 2
    //because of the top and bottom
    for (int row=0; row<boxSize-2; row++)
    {
        drawEdges(boxSize);
    }

    //draw the bottom line
    drawLine(boxSize);
}
```

## Section 6

### Example: Finding Prime Numbers

# Finding Prime Numbers

- Let's do a small program to find prime numbers



# Prime Number Background

- A number is a prime number (or *prime*)
- If there's no number that divides into it evenly

For example:

- 14 is divided by 7 evenly - 14 is not *prime*
- 7 is only divided by 1 and 7 - it is *prime*

# Prime Number Test

The test is:

Given a number  $n$ , is there a number  $a$  where  $n \% a = 0$ ?

Where  $\%$  is the mod operator which gives a remainder

Let's build a method to do this

- **Name:** isPrime
- **Parameters:** An int  $n$
- **Returns:** A boolean whether  $n$  is prime or not

# isPrime Method

```
boolean isPrime = true;
//start from two and go to the number
for (int a = 2; a < x; a++)
{
    //if the remainder of dividing x by i is 0,
    //then x cannot be a prime
    if (x % a == 0)
    {
        isPrime = false;
    }
    //don't put an else here!
}
return isPrime;
```

- Note that we assume that  $x$  is prime, and we change the boolean to false when we find a counter-example
- Don't put an else here! Once  $x$  is not a prime, we can't say it is a prime again
- We can also make this faster by only going up to the square root of  $n$

# findPrimes Method

This method will loop from 2 to the parameter, and report all primes found

```
//find all the primes up to the max parameter
public static void findPrimes(int max)
{
    for (int i=2; i < max; i++)
    {
        boolean iIsPrime = isPrime(i);

        if (iIsPrime)
        {
            System.out.println("Prime number: " + i);
        }
    }
}
```

```
Prime number: 2
Prime number: 3
Prime number: 5
Prime number: 7
Prime number: 11
Prime number: 13
Prime number: 17
Prime number: 19
```

## Section 7

### Example: Collatz Conjecture

- Let's look at an math problem which is unsolved
- It's called the *Collatz Conjecture*
  - Conjecture is another word for theory

“this is an extraordinarily difficult problem, completely out of reach of present day mathematics.”  
Don't worry, it's easy to explain!

The conjecture states:

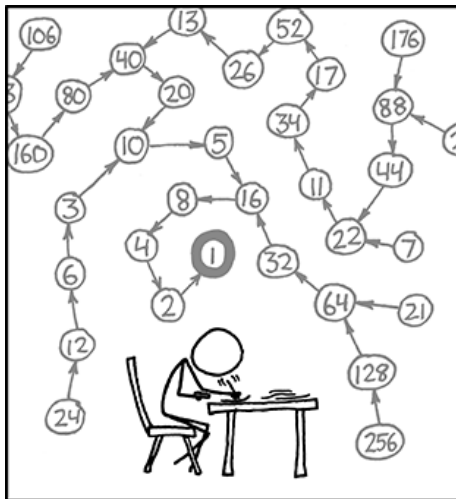
- Start with a number  $n$
- If  $n$  is even, divide  $n$  by 2
- If  $n$  is odd, multiply  $n$  by 3 and add 1
- **No matter what  $n$  you start with, eventually a 1 is produced**

Examples:

- Starting with  $n = 12$  gives the sequence 12, 6, 3, 10, 5, 16, 8, 4, 2, 1
- $n = 19$  takes a longer time to end: 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

So is there any sequence that doesn't end in a 1?

# Collatz Conjecture XKCD



“The Collatz Conjecture states that if you pick a number,

and if it's even divide it by two and if it's odd, multiply it by three and add one,

and you repeat this procedure long enough,

eventually your friends will stop calling to see if you want to hang out.”



- We won't solve this hard problem
- But let's write a method that generates the Collatz sequence for a particular number
- **Method Name:** `collatz`
- **Parameter:** `int n`
- **Return Type:** `int`
- **Description:** Takes in a `int n`, prints out the numbers in the Collatz sequence, and returns an `int` which is the length of that sequence
- **Example:** `collatz(12)` prints out 12, 6, 3, 10, 5, 16, 8, 4, 2, 1, and returns 10

# Planning Out Our Method

- So we start with our number  $n$
- Until  $n$  becomes one
  - If  $n$  is even
    - Divide it by two
  - If  $n$  is odd
    - Multiply it by three and add one
  - Print each step in this sequence
  - Count how many items are in this sequence
- Return the count of items

- Again start with just enough to compile

```
public static int collatz(int n)
{
    //the length of the sequence
    int seqLength = 1;
    System.out.print(n);

    System.out.println();
    return seqLength;
}
```

# Collatz Conjecture

- Hard to divide this up into smaller steps
- This produces the next step in the sequence

```
public static int collatz(int n)
{
    //the length of the sequence
    int seqLength = 1;
    System.out.print(n);

    if (n % 2 == 0){ //if n is even
        n = n / 2; //divide n by two
    }else{ //if n is odd
        //triple n and add one
        n = 3*n + 1;
    }
    //print out the new value of n
    System.out.print(", " + n);
    seqLength++;

    System.out.println();
    return seqLength;
}
```

# Collatz Conjecture

- Add the while loop to test if  $n$  is one

```
public static int collatz(int n)
{
    //the length of the sequence
    int seqLength = 1;
    System.out.print(n);

    while (n != 1){ //while n is not 1
        if (n % 2 == 0){ //if n is even
            n = n / 2; //divide n by two
        } else { //if n is odd
            //triple n and add one
            n = 3*n + 1;
        }
        //print out the new value of n
        System.out.print(", " + n);
        seqLength++;
    }

    System.out.println();
    return seqLength;
}
```

- Let's add another method
- Now that we can get the length of the Collatz sequence for a number,
- **Which number between 1 and 100 has the longest sequence?**
- Method Name: `maxLength`
- Parameter: `int topNum`
- Return Type: `int`
- Description: Given the parameter `int topNum`, this method returns the `int` number between 1 and `topNum` which has the longest Collatz sequence

# Collatz Conjecture

```
public static int maxLength(int topNum)
{
    //save the longest sequence length
    //and the number who's sequence it is
    int numberWithLongestSequence = 1;
    int longestSequenceLength = 1;

    //loop through from 1 to topNum
    for (int i=1; i <= topNum; i++)
    {
        //get the length
        int length = collatz(i);

        //if the length is bigger than
        //the stored length,
        //update the longest length
        //and the longest number
        if (length > longestSequenceLength){
            longestSequenceLength = length;
            numberWithLongestSequence = i;
        }
    }
    return numberWithLongestSequence;
}
```

- The main method to call the other two methods

```
public static void main(String[] args)
{
    int longest = maxLength(100);
    System.out.println("The num with longest sequence is: " + longest);
    System.out.println("The length is: " + collatz(longest));
}
```