Methods, Parameters, and Scope

Lecture 6

CS106A, Summer 2019
Sarai Gould && Laura Cruz-Albrecht

Announcements

- Assignment 1 is due at **10am tomorrow morning.**
  - **Practice submitting today to make sure you know how to do it!**

- No lecture or sections Thursday for 4th of July.
  - If your section in cancelled, please try to attend a Wednesday section or Friday's section (11:30am in Skilling Auditorium)
- No LaIR Wednesday, July 3rd due to Holiday

Announcements

- **Quick Note:** If you joined the class late, you will receive your lecture assignments for Lecture Feedback on Monday, July 8th.

Duke, the Java mascot!

Hi! My name is Duke!

Plan for Today

- Review: Shorthand Operators, Constants, If, While, For
- Remember Methods?
- Making a Sandwich
- A Variable Love Story

Review: Shorthand Operators

Shorthand:

```
// +, -, /, *, % any value
variable += value;
x -= 3;
y /= 5;
z *= someValue;
k %= 10;

// add or subtract exactly 1
x++;
y--;
```

Equivalent Longer Version:

```
variable = variable + value;
x = x - 3;
y = y / 5;
z = z * someValue;
k = k % 10;

x = x + 1;
y = y - 1;
```

Review: Constants

```
public class ReceiptForFive extends ConsoleProgram {
    private static final double TAX_RATE = 0.08;
    private static final double TIP_RATE = 0.2;
    public void run() {
        double subtotal1 = readDouble("Meal cost? $");
        double tax1 = subtotal * TAX_RATE;
        double tip1 = subtotal * TIP_RATE;
        double total1 = subtotal1 + tax1 + tip1;
        println("Total for person 1: $" + total1);
        ...
        double subtotal5 = readDouble("Meal cost? $");
        double tax5 = subtotal * TAX_RATE;
        double tip5 = subtotal * TIP_RATE;
        double total5 = subtotal1 + tax1 + tip1;
        println("Total for person 5: $" + total5);
    }
}
```

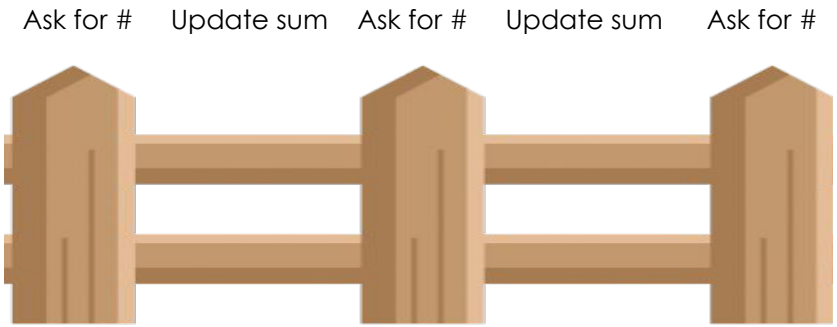
much better

Review: Relational Operators

Operator	Meaning	Example	Value
==	equals	`1 + 1 == 2`	`true`
!=	does not equal	`3.2 != 2.5`	`true`
<	less than	`10 < 5`	`false`
>	greater than	`10 > 5`	`true`
<=	less than or equal to	`126 <= 100`	`false`
>=	greater than or equal to	`5.0 >= 5.0`	`true`

* all have equal precedence

Review: Sentinel Loops




```
// fencepost problem!
// ask for number - post
// add number to sum - fence

int sum = 0;
int num = readInt("Enter a number: ");
while (num != -1) {
    sum += num;
    num = readInt("Enter a number: ");
}
println("Sum is " + sum);
```

Review: Using the For Loop Variable

```
// Adds up the numbers 1 through 42
int sum = 0;
for (int i = 1; i <= 42; i++) {
    sum += i;
}
println("Sum is " + sum);
```

???

I have a question...

Making Sandwiches

Would someone make me a sandwich, please?

Making Sandwiches

Let's make a method that will make Duke a sandwich!

Methods

A **method** is a set of new instructions we've created!

Example of a comment explaining Pre and Post conditions.

```
/* Method description.
* Pre: What we assume is true beforehand.
* Post: What we promise is true afterwards.
*/
private void nameOfMethod(){

     // commands go in here!

}
```

Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich
*/
private void makeASandwich(){


}
```

Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich
*/
private void makeASandwich(){


}
```

Pretend we've added a new primitive variable type called "**food**".

Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich
*/
private void makeASandwich(food bread, food veg, food protein, food spread){


}
```

By using **parameters** we can tell our method what the ingredients for our sandwich are!

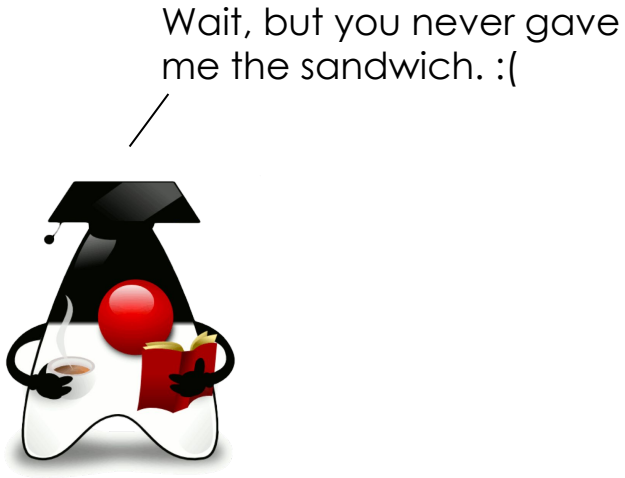
Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich.
*/
private void makeASandwich(food bread, food veg, food protein, food spread){

}
```

Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich.
*/
private void makeASandwich(food bread, food veg, food protein, food spread){
     food sandwich = bread + veg + protein + spread + bread;

}
```

Making Sandwiches



Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich.
*/
private void makeASandwich(food bread, food veg, food protein, food spread){
     food sandwich = bread + veg + protein + spread + bread;

}
```

Making Sandwiches

```
/* We will make Java a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich
*/
private void makeASandwich(food bread, food veg, food protein, food spread){
     food sandwich = bread + veg + protein + spread + bread;

     return sandwich;
}
```

By using a **return statement** a method can give back, or **return**, information to the rest of your code!

Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich
*/
private food makeASandwich(food bread, food veg, food protein, food spread){
     food sandwich = bread + veg + protein + spread + bread;

     return sandwich;
}
```

Since our method is returning information, it's no longer **void**. We change void to the **type of data we are returning!**

Making Sandwiches

```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich.
*/
private food makeASandwich(food bread, food veg, food protein, food spread){
     food sandwich = bread + veg + protein + spread + bread;

     return sandwich;
}
```

Making Sandwiches

Methods w/ Parameters and Return Statements

```
/* We will do x, y, and z actions and return methodType data
* Pre: What we assume is true beforehand.
* Post: What we promise is true afterwards.
*/
visibility methodType methodName(paramType paramName){

     commands;

     return dataThatIsMethodType;
}
```

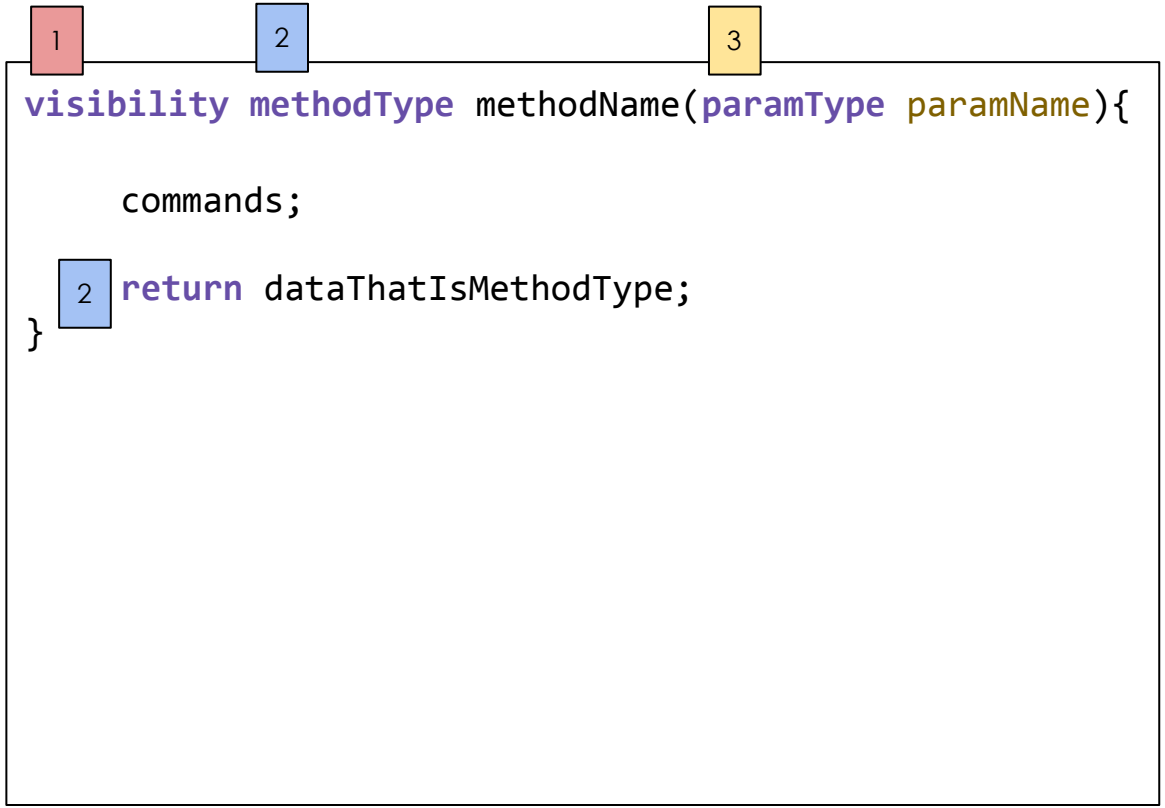
Methods w/ Parameters and Return Statements

1 **Visibility:**
Usually **`public`** or **`private`**

Types:

2 **`methodType`** is the type of data returned by the method. This is **`void`** when the method doesn't return any data.

3 **`paramType`** is the type of data that particular parameter is. Parameters are information the method needs in order for it to work!



Explained Another Way...

A method is like a recipe. You give it a name and tell us what type of thing it will make!

The parameters are the ingredients you need to make it.

You have to return something of the same type that the recipe promised.

If something's the wrong type, you're missing ingredients, the recipe promised a cake, but you return spaghetti… something is seriously wrong with your recipe!

Explained Another Way...

A method is like a recipe. You give it a name and tell us what type of thing it will make!

The parameters are the i[illegible] to make it.

You have to return s[illegible] the recipe promised.

If something's the wrong type, y[illegible]g ingredients, the recipe promised a cake, but you return spaghetti… something is seriously wrong with your recipe!

A Few Examples

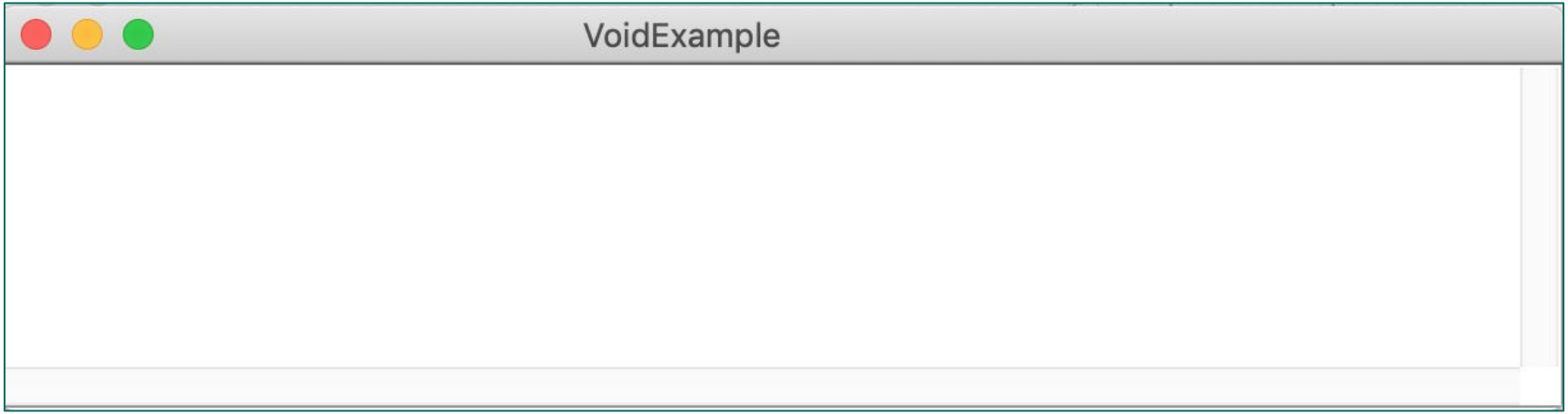
```
private void printHello(){
     println(“hello world”);
     println(“I’m a void method, so I don’t return any data!”);
     println(“I just do an action.”);
}
```

A Few Examples

```
private void printHello(){
     println("hello world");
     println("I'm a void method, so I don't return any data!");
     println("I just do an action.");
}


private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

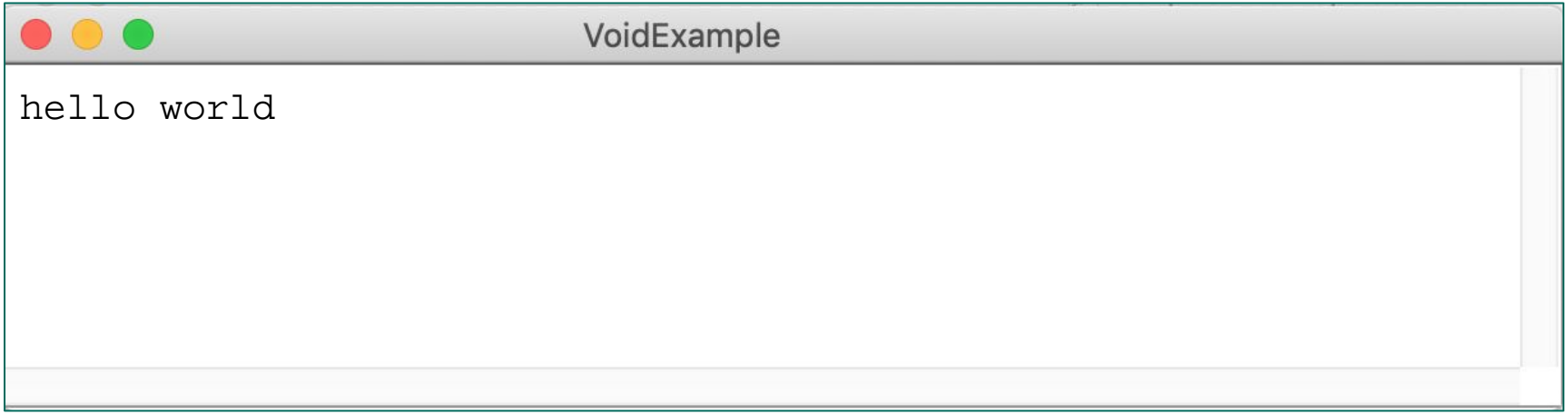
A Few Examples

```
private void printHello(){
     println("hello world");
     println("I'm a void method, so I don't return any data!");
     println("I just do an action.");
}
```



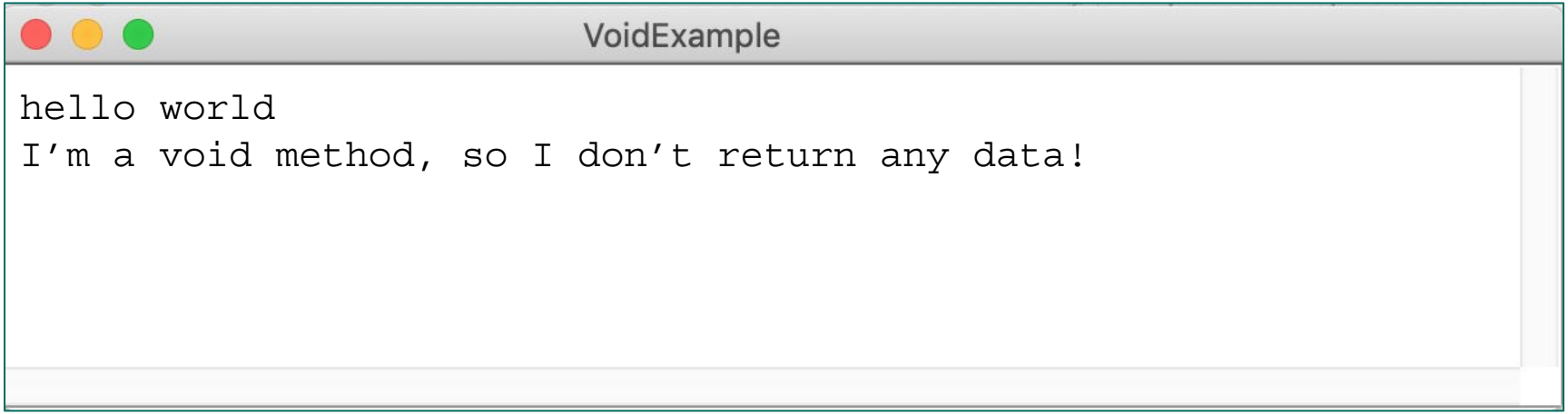
A Few Examples

```
private void printHello(){
     println("hello world");
     println("I'm a void method, so I don't return any data!");
     println("I just do an action.");
}
```



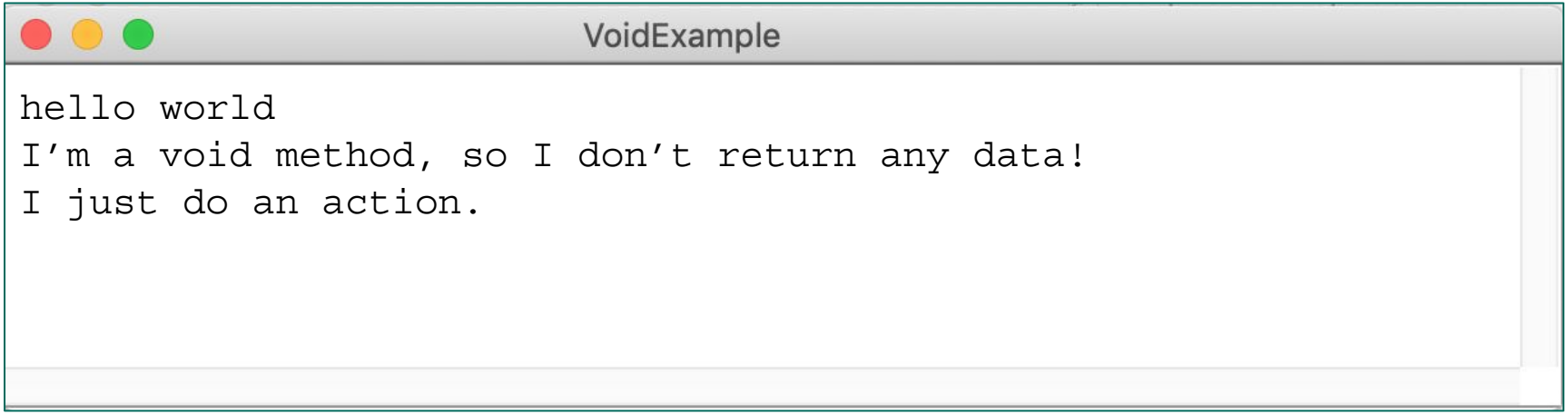
A Few Examples

```
private void printHello(){
     println("hello world");
     println("I'm a void method, so I don't return any data!");
     println("I just do an action.");
}
```



A Few Examples

```
private void printHello(){
     println("hello world");
     println("I'm a void method, so I don't return any data!");
     println("I just do an action.");
}
```

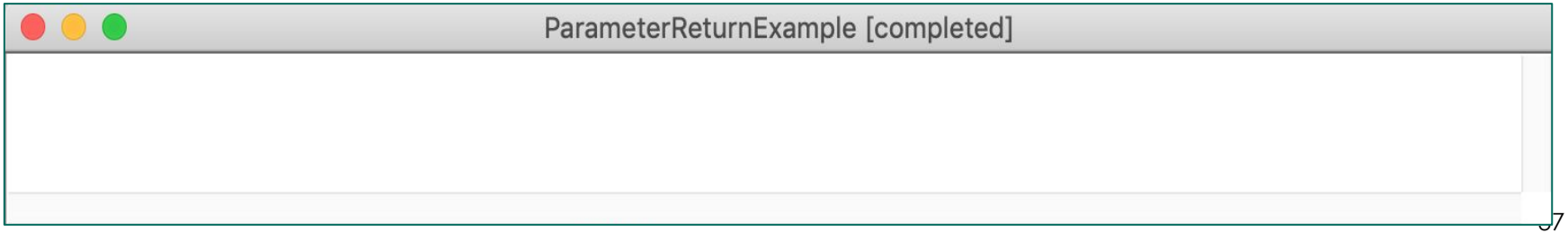


A Few Examples

```
private void printHello(){
     println("hello world");
     println("I'm a void method, so I don't return any data!");
     println("I just do an action.");
}


private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

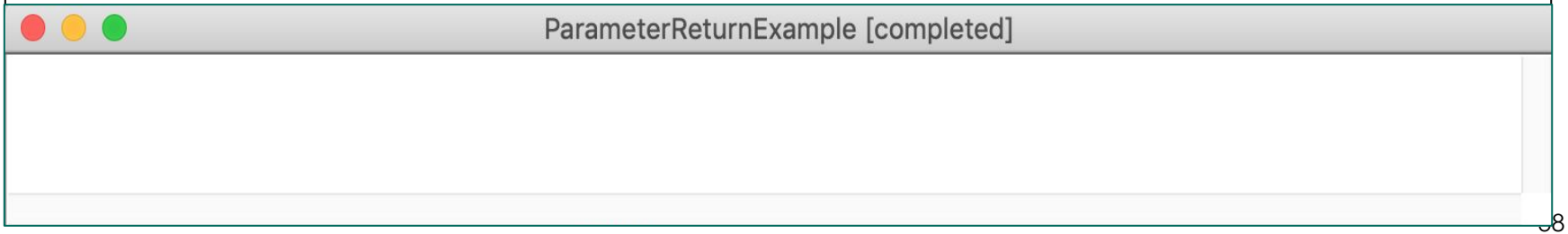
A Few Examples

```
private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```



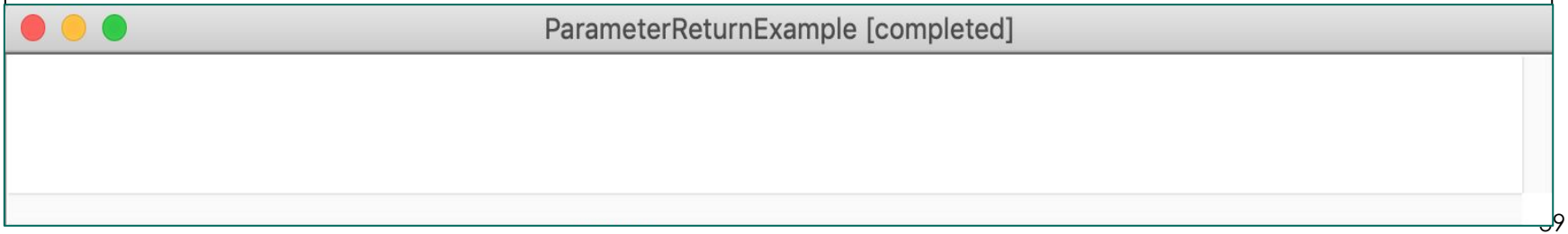
A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```



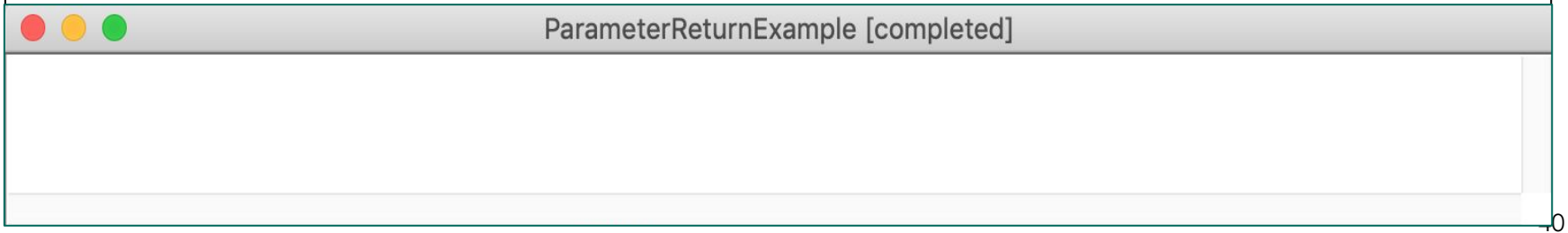
A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```



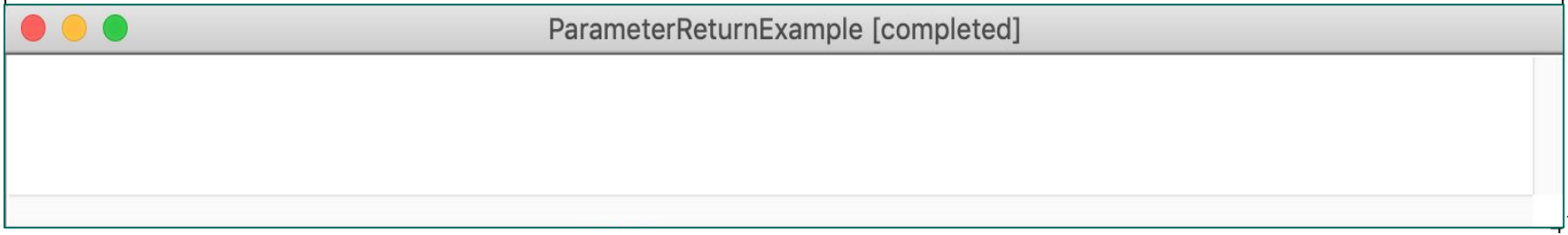
A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```



A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```



A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

5.0

num1

ParameterReturnExample [completed]

A Few Examples

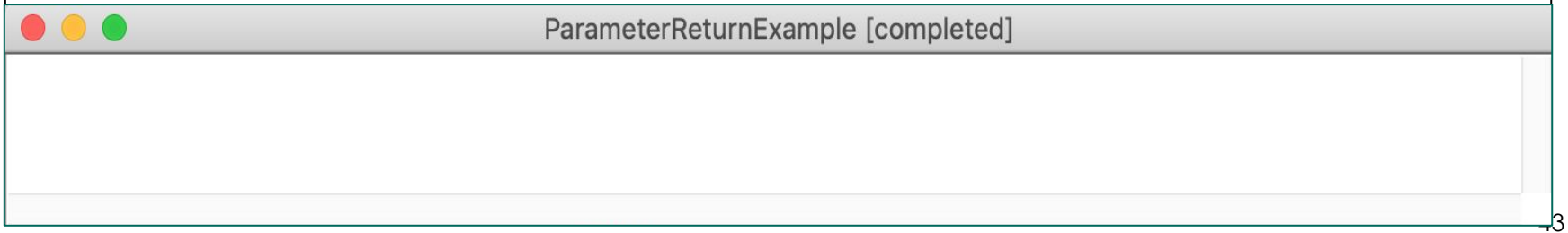
```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

5.0

num1

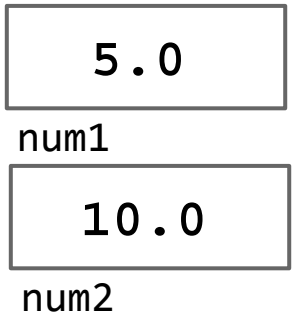
10.0

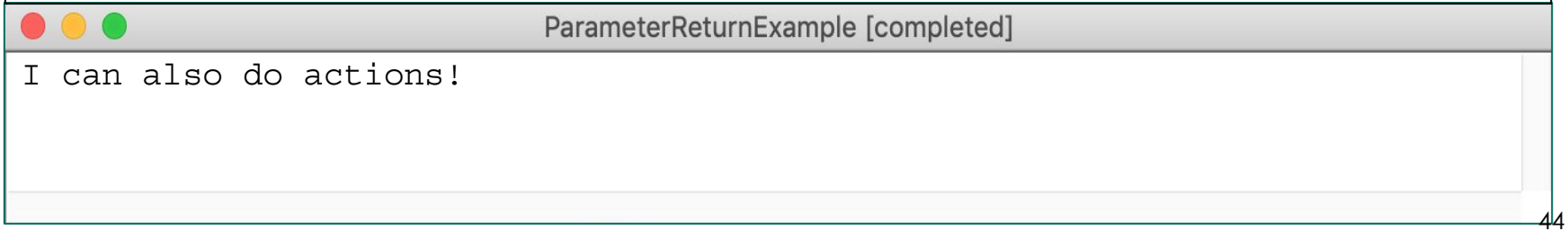
num2



A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```





A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

5.0

num1

10.0

num2

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

5.0

num1

10.0

num2

5.0 + 10.0

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2;
}
```

5.0
num1

10.0
num2

15.0
sum

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2; // This will equal 7.5
}
```

5.0
num1

10.0
num2

15.0
sum

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2; // This will equal 7.5
}
```

5.0
num1

10.0
num2

15.0
sum

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2; // This will equal 7.5
}
```

7.5

calculateAverage(5, 10)

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2; // This will equal 7.5
}
```

7.5

average

ParameterReturnExample [completed]

```
I can also do actions!
I also HAVE to return a double because that's the method type!
```

A Few Examples

```
public void run(){
     double average = calculateAverage(5, 10);
     println("The average is: " + average);
}

private double calculateAverage(double num1, double num2){
     println("I can also do actions!");
     println("I also HAVE to return a double because that's the method type!");

     double sum = num1 + num2;
     return sum / 2; // This will equal 7.5
}
```

7.5

average

ParameterReturnExample [completed]

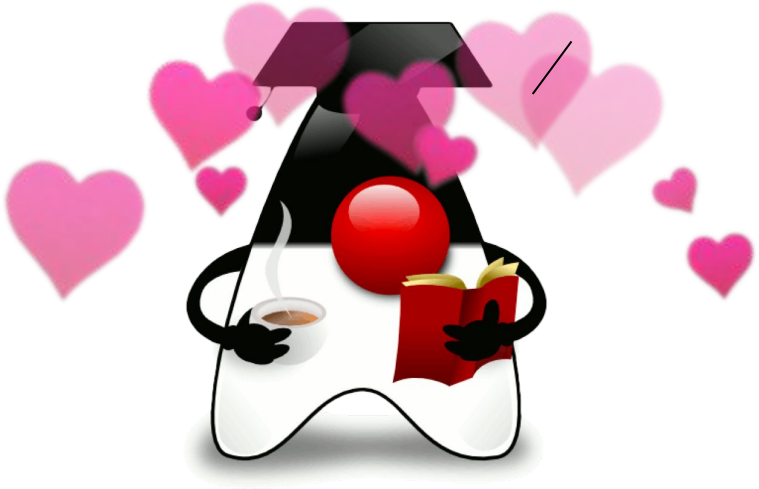
```
I can also do actions!
I also HAVE to return a double because that's the method type!
The average is 7.5
```

Now, Time for a Story

Now, Time for a Story

It's time to hear a **Variable Love Story.**

I love Love Stories...



A Variable Love Story

romeo was lonely and looking for love...

```
public void run(){
     int romeo = 18;

}
```

A Variable Love Story

What's this? That sounds great!

```
public void run(){
     int romeo = 18;
     findRomance();
}
```

*Like a dating site, but for variables

A Variable Love Story

"But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun."
~ *romeo, probably*

```
public void run(){
     int romeo = 18;
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story

```
public void run(){
     int romeo = 18;
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

romeo cannot be resolved to a variable????

A Variable Love Story

romeo cannot be resolved to a variable????

```
public void run(){
     int romeo = 18;
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story

What about `trueLove`???

```
public void run(){
     int romeo = 18;
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story

`romeo` only exists between those two brackets!

```
public void run(){
     int romeo = 18;
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story

`romeo` only exists between those two brackets!

Once we enter `findRomance`, `romeo` essentially disappears.

How can we make `romeo` exist inside of `findRomance`?

```
public void run(){
     int romeo = 18;
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story

What about using parameters?

A Variable Love Story

We'll add a parameter to `findRomance`!

This means `findRomance` needs more information (aka an extra ingredient) in order to work.

```
public void run(){
    int romeo = 18;
    findRomance();
}

private void findRomance(int r){
    int juliet = 13;

    int trueLove = romeo + juliet;
}
```

A Variable Love Story

At the beginning of the code, we'll pass `romeo` into our `findRomance` method.

Now, our parameter `r` will be equal to what `romeo` was equal to!

So, `r + juliet` is essentially equal to `romeo + juliet`.

```
public void run(){
    int romeo = 18;
    findRomance(romeo);
}

private void findRomance(int r){
    int juliet = 13;

    int trueLove = r + juliet;
}
```

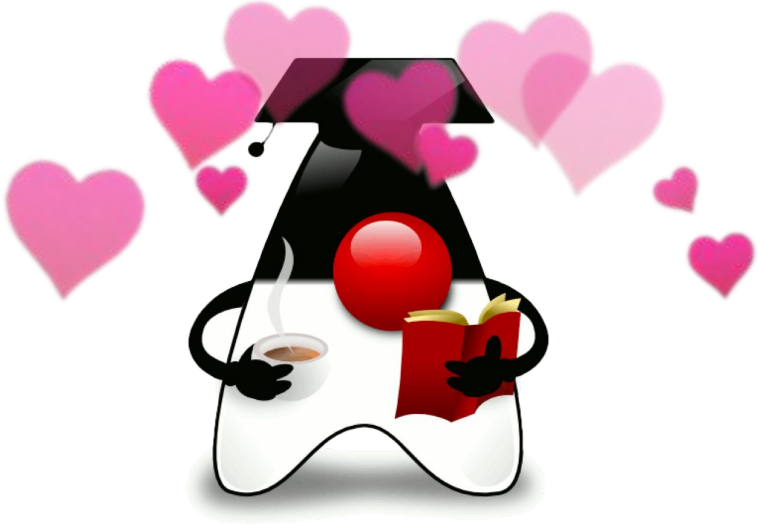
A Variable Love Story

What does this mean?

```
public void run(){
     int romeo = 18;
     findRomance(romeo);
}

private void findRomance(int r){
     int juliet = 13;

     int trueLove = r + juliet;
}
```

A Variable Love Story

`trueLove` at last!



*We do not condone relationships of 13 and 18 year olds. Blame Shakespeare.

```
public void run(){
     int romeo = 18;
     findRomance(romeo);
}

private void findRomance(int r){
     int juliet = 13;

     int trueLove = r + juliet;
}
```

A Variable Love Story: Scope

Scope is the idea that variables only exist inside a certain block of code.

In Java, a variable is **born when it is declared.**

A variable **terminates when it hits the ending bracket** of the code block in which it was declared.

```
public void run(){
     int newVariable = 0; // I am born!
     ...
     ...
     ...
} // I hope I haven't bored you. Goodbye.
```

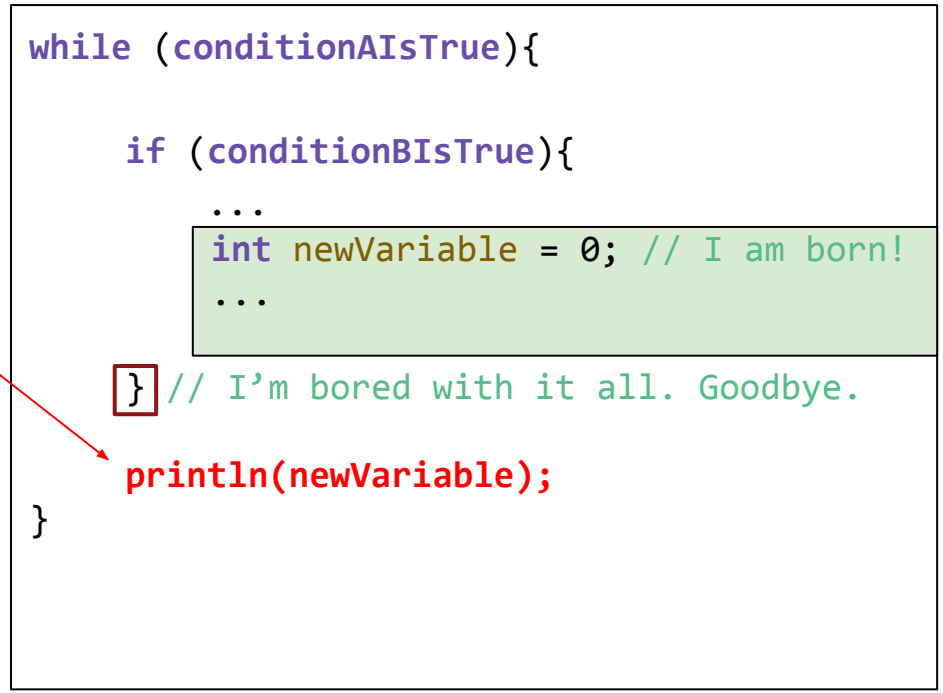
A Variable Love Story: Scope

The variable only exists from its declaration to the end of its current code block.

```
while (conditionAIsTrue){

     if (conditionBIsTrue){
          ...
          int newVariable = 0; // I am born!
          ...

     } // I'm bored with it all. Goodbye.

}
```

A Variable Love Story: Scope

This won't work!



```
while (conditionAIsTrue){

    if (conditionBIsTrue){
        ...
        int newVariable = 0; // I am born!
        ...

    } // I'm bored with it all. Goodbye.

    println(newVariable);
}
```

A Variable Love Story: Scope

It doesn't exist before it's declared, and it doesn't exist outside of its current code block!

```
while (conditionAIsTrue){

     if (conditionBIsTrue){
          ...
          int newVariable = 0; // I am born!
          ...

     } // I'm bored with it all. Goodbye.

     println(newVariable);

}
```

A Variable Love Story: Scope

Why can't we just declare romeo outside of any brackets?

```
int romeo = 18;

public void run(){
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story: Scope

Why can't we just declare romeo outside of any brackets?

1. It's harder to understand.

```
int romeo = 18;

public void run(){
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story: Scope

Why can't we just declare `romeo` outside of any brackets?

1. It's harder to understand.
2. More likely to accidentally introduce bugs.

```
int romeo = 18;

public void run(){
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story: Scope

Why can't we just declare `romeo` outside of any brackets?

1. It's harder to understand.
2. More likely to accidentally introduce bugs.
3. We don't get rid of `romeo` when we're done with `romeo`.

```
int romeo = 18;

public void run(){
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story: Scope

Don't do this!

```
int romeo = 18;

public void run(){
     findRomance();
}

private void findRomance(){
     int juliet = 13;

     int trueLove = romeo + juliet;
}
```

A Variable Love Story: Scope

Do this!

It's easier to read and debug, and we remove variables from memory when we're done with them!

```
public void run(){
     int romeo = 18;
     findRomance(romeo);
}

private void findRomance(int r){
     int juliet = 13;

     int trueLove = r + juliet;
}
```

Plan for Today

- Review: Shorthand Operators, Constants, If, While, For
- Remember Methods?
- Making a Sandwich
- A Variable Love Story

Reminders

- Assignment 1 is due at **10am tomorrow morning.**
  - **Practice submitting today to make sure you know how to do it!**

- No lecture or sections Thursday for 4th of July.
  - If your section in cancelled, please try to attend a Wednesday section or Friday's section (11:30am in Skilling Auditorium)
- No LaIR Wednesday, July 3rd due to Holiday