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;
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 = subtotal1 * TAX_RATE;
        double tip1 = subtotal1 * TIP_RATE;
        double total1 = subtotal1 + tax1 + tip1;
        println("Total for person 1: ");
        ...
        double subtotal5 = readDouble("Meal cost? ");
        double tax5 = subtotal5 * TAX_RATE;
        double tip5 = subtotal5 * TIP_RATE;
        double total5 = subtotal5 + tax5 + tip5;
        println("Total for person 5: ");
    }
}
### Review: Relational Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>1 + 1 == 2</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>3.2 != 2.5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>10 &lt; 5</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>10 &gt; 5</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>126 &lt;= 100</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>5.0 &gt;= 5.0</td>
<td>true</td>
</tr>
</tbody>
</table>

* 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);
// 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...
Would someone make me a sandwich, please?
Let’s make a method that will make Duke a sandwich!
A method is a set of new instructions we’ve created!

Example of a comment explaining Pre and Post conditions.

```java
/* Method description.
 * Pre: What we assume is true beforehand.
 * Post: What we promise is true afterwards.
 */
private void nameOfMethod()
{
    // commands go in here!
}
```
/* We will make Duke a sandwich.
* Pre: n/a
* Post: We will have a completed sandwich
*/

private void makeASandwich(){
/* We will make Duke a 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){
}
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){
}

/* 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;
}


Wait, but you never gave me the sandwich. :(
/ * 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!
/* 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;
}
/* We will make Duke a sandwich. */
* Pre: n/a
* Post: We will have a completed sandwich.
*/

```java
private food makeASandwich(food bread, food veg, food protein, food spread)
{
    food sandwich = bread + veg + protein + spread + bread;
    return sandwich;
}
```
Making Sandwiches

Thanks! 🥪
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;
}
Visibility: Usually **public** or **private**

**Types:**
- `methodType` is the type of data returned by the method. This is **void** when the method doesn't return any data.
- `paramType` is the type of data that particular parameter is. Parameters are information the method needs in order for it to work!
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 ingredients to make it.

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

If something’s the wrong type, you’re using ingredients, the recipe promised a cake, but you return spaghetti… something is seriously wrong with your recipe!
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 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;
}
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 void printHello(){
    println("hello world");
    println("I’m a void method, so I don’t return any data!");
    println("I just do an action.");
}

hello world
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.");
}

hello world
I’m a void method, so I don’t return any data!
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

```java
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;
}
```
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

```java
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;
}
```
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;
}
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;
}
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;
}
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!"toJson);
    println("I also HAVE to return a double because that’s the method type!");
    double sum = num1 + num2;
    return sum / 2;
}
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

```java
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;
}
```

I can also do actions!
A Few Examples

```java
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;
}
```
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;
}

I can also do actions!
I also HAVE to return a double because that’s the method type!
A Few Examples

```java
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;
}
```

I can also do actions!
I also HAVE to return a double because that’s the method type!
A Few Examples

```java
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
}
```

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

```java
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
}
```

I can also do actions!
I also HAVE to return a double because that’s the method type!
A Few Examples

```java
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
}
```

I can also do actions!
I also HAVE to return a double because that’s the method type!
A Few Examples

```java
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
}
```

I can also do actions!
I also HAVE to return a double because that’s the method type!
A Few Examples

```java
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
}
```

- 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
It’s time to hear a **Variable Love Story**.
A Variable Love Story

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

**romeo** was lonely and looking for love...
A Variable Love Story

What’s this? That sounds great!

*Like a dating site, but for variables

```java
public void run(){
    int romeo = 18;
    findRomance();
}
```
A Variable Love Story

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

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

private void findRomance(){
    int juliet = 13;

    int trueLove = romeo + juliet;
}
```
A Variable Love Story

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

private void findRomance()
{
    int juliet = 13;

    int trueLove = romeo + juliet;
}

romeo cannot be resolved to a variable????
```
romeo cannot be resolved to a variable????

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

private void findRomance(){
    int juliet = 13;

    int trueLove = romeo + juliet;
}
```
What about trueLove???

```java
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!
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?
A Variable Love Story

What about using parameters?
We’ll add a parameter to `findRomance`!

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

What does this mean?

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

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

    int trueLove = r + juliet;
}
```
public void run(){
    int romeo = 18;
    findRomance(romeo);
}

private void findRomance(int r){
    int juliet = 13;
    int trueLove = r + juliet;
}
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.

```java
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.

```java
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!

```java
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!

```java
while (conditionAIsTrue){
    if (conditionBIsTrue){
        ...
        int newVariable = 0; // I am born!
        ...
    }
    // I’m bored with it all. Goodbye.
}
println(newVariable);
```
Why can’t we just declare `romeo` outside of any brackets?

```java
int romeo = 18;

public void run(){
    findRomance();
}

private void findRomance(){
    int juliet = 13;
    int trueLove = romeo + juliet;
}
```
Why can’t we just declare `romeo` outside of any brackets?

1. It’s harder to understand.
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.

```java
int romeo = 18;

public void run(){
    findRomance();
}

private void findRomance(){
    int juliet = 13;

    int trueLove = romeo + juliet;
}
```
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`.

```java
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!

```java
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!

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

private void findRomance(int r){
   int juliet = 13;
   int trueLove = r + juliet;
}
```

Do this!
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