Lab: Parameters

The Slope Formula

You are going to write a method that takes in 4 **doubles** as parameters. These parameters represent two sets of coordinate points labeled as `x1`, `y1`, `x2`, and `y2`. The method will then use these points to calculate the slope that their line creates and then prints that slope to the user.

Method Header

First, we need to set up the method header. Start off with `public static` followed by our return type. Since the result is simply printing the slope, we will use `void` as the return type. Additionally, we'll name the method `getSlope()`.

```
public static void getSlope() {
}
```

Parameters

The method should take 4 **doubles** as parameters named `x1`, `y1`, `x2`, and `y2`.

```
public static void getSlope(double x1, double y1, double x2,
        double y2) {
}
```

Printing the Slope

The final step is to print the slope, but we'll need the slope formula in order to do that. The slope formula is defined as `(y2 - y1) / (x2 - x1)`.

```
System.out.println((y2 - y1) / (x2 - x1));
```

Testing the Method

In order to use a method, you'll need to call it by specifying its name within the `main()` method. Note that the method requires parameters so we'll need to provide some arguments in order for the method to work properly. Let's use the points `(3, 2)` and `(5, 6)` as our coordinates which correspond to `(x1, y1)` and `(x2, y2)` respectively.

```
public static void main(String args[]) {
  getSlope(3, 2, 5, 6);
}
```

The `getSlope()` method will apply the slope formula `(6 - 2) / (5 - 3)` and print the result `2.0` to the user. Make sure to also include documentation so that other users can understand your method.

▼ **Code**

```
/**
 * This method prints the slope between two sets
 * of coordinate points
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return No return value
 */
public static void getSlope(double x1, double y1, double x2,
      double y2) {
  System.out.println((y2 - y1) / (x2 - x1));
}

public static void main(String args[]) {
  getSlope(3, 2, 5, 6);
}
```

Lab: Variable Scope

Local and Global Variables

For this lab, we are going to be adding local and global variables to our previously created `getSlope()` method. Remember that **global** variables exist *outside* of methods while **local** variables exist *inside* methods. Depending on how you declare your local and global variables, they will behave differently per situation.

Global Variables

First, let's add some global variables to our program.

```
double input1;
double input2;
double input3;
double input4;
```

The getSlope() Method

As from before, the method will still take 4 **doubles** as parameters named `x1`, `y1`, `x2`, and `y2`. However, we're going to implement two different calculations within the method. Specifically, we are going to calculate the difference between the **y** coordinates first, then calculate the difference between the **x** coordinates. These calculations will then be assigned to **local** variables called `yChange` and `xChange`. Finally, the method will print the quotient between `yChange` and `xChange`, which is also the slope itself.

```
public static void getSlope(double x1, double y1, double x2,
        double y2) {
  double yChange = y2 - y1;
  double xChange = x2 - x1;
  System.out.println(yChange / xChange);
}
```

Testing the Method

To make things more dynamic, we'll actually make use of a `scanner` within the `main()` method. The `scanner` will take in inputs from the user and assign them to our 4 global variables `input1`, `input2`, `input3`, and `input4`. These inputs will later correspond to `x1`, `y1`, `x2`, and `y2`. Having the scanner enables the user to decide what the coordinate points will be.

```
public static void main(String args[]) {
  Scanner input = new Scanner (System.in);
  System.out.println("Enter first x-coordinate: ");
  input1 = input.nextDouble();
  System.out.println("Enter first y-coordinate: ");
  input2 = input.nextDouble();
  System.out.println("Enter second x-coordinate: ");
  input3 = input.nextDouble();
  System.out.println("Enter second y-coordinate: ");
  input4 = input.nextDouble();
  input.close();

  getSlope(input1, input2, input3, input4);
}
```

Before we can test the code, however, we need to include one important keyword or the code will not compile. That important keyword is `static`. Recall that the variables `input1`, `input2`, `input3`, and `input4` were declared previously as global variables. Now we are trying to manipulate those variables within `main()`. We cannot do this however because static methods cannot access non-static variables. Therefore, we must include `static` before all of the variables we had declared globally.

```
static double input1;
static double input2;
static double input3;
static double input4;
```

Side note: In Java, *static* local variables do not exist.

You'll notice that you have access to the Terminal which you will use to input any coordinate points you want. If you enter `3`, `2`, `5`, and `6` respectively, you should get `2.0`. Click the `COMPILE` button first to compile the program, and then click `RUN` to run the program and enter inputs.

▼ **Code**

```java
static double input1; //global
static double input2; //global
static double input3; //global
static double input4; //global

/**
 * This method prints the slope between two sets
 * of coordinate points by calculating their coordinate
 * changes separately
 *
 * @param x1 A double for first x-coordinate
 * @param y1 A double for first y-coordinate
 * @param x2 A double for second x-coordinate
 * @param y2 A double for second y-coordinate
 * @return No return value
 */
public static void getSlope(double x1, double y1, double x2,
      double y2) {
  double yChange = y2 - y1; //local
  double xChange = x2 - x1; //local
  System.out.println(yChange / xChange);
}

public static void main(String args[]) {
  Scanner input = new Scanner (System.in);
  System.out.println("Enter first x-coordinate: ");
  input1 = input.nextDouble();
  System.out.println("Enter first y-coordinate: ");
  input2 = input.nextDouble();
  System.out.println("Enter second x-coordinate: ");
  input3 = input.nextDouble();
  System.out.println("Enter second y-coordinate: ");
  input4 = input.nextDouble();
  input.close();

  getSlope(input1, input2, input3, input4);
}
```

info

Program Flow

After the program is initiated, the global variables will be created first. Next, `main()` will run. Although commonly written last, `main()` will always be the first method to run by default in Java. The lines of code within `main()` will be executed in the order of their appearance.

Lab: Return Values

Returning a Value

When the result of a method is simply a print statement, it is considered to be a `void` method. `void` methods do not have a return type, meaning they do not return data back to the user. To return data, use the keyword `return` followed by the data. Note that methods with `return` must be declared with the same data type as the data that they return. For example, a method that returns a `double` must be declared in the header as a `double` method.

```
public static double getSlope(double x1, double y1,
                              double x2, double y2) { //replace
        void with double
  double yChange = y2 - y1;
  double xChange = x2 - x1;
  return yChange / xChange; //returns a double
}
```

Modifying the Return Value

Notice that our method returns a single double, which is nice but not extremely helpful when it comes to determining *rise* and *run* for slopes (`rise / run`). Let's say we want instead to express the slope in the `rise / run` format. `rise` is the change in `y` values and `run` is the change in `x` values. Unfortunately, we can't simply do `return yChange + " / " + xChange`. Why? Because the `" / "` is a string which is not compatible with the current `return` value of `double`. One way around this is to convert the doubles into strings. Doing so will force us to change our `double` method into a `String` method.

```
public static String getSlope(double x1, double y1,
                              double x2, double y2) { //replace
        double with string
  double yChange = y2 - y1;
  double xChange = x2 - x1;
  return yChange + " / " + xChange; //returns a string
}
```

Notice how we did not have to do any conversion. This is due to the fact that we have `+ " / " +` with our return statement. When Java encounters concatenation that involves a string, it will automatically convert everything within that statement into a string.

Completing the Program

Now just copy over the rest of the program that we had previously written.

```
static double input1;
static double input2;
static double input3;
static double input4;

/**
 * This method returns the slope between two sets
 * of coordinate points by calculating their coordinate
 * changes separately
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A String expression of the slope in rise / run format
 */
public static String getSlope(double x1, double y1,
                              double x2, double y2) {
  double yChange = y2 - y1;
  double xChange = x2 - x1;
  return yChange + " / " + xChange;
}

public static void main(String args[]) {
  Scanner input = new Scanner (System.in);
  System.out.println("Enter first x coordinate: ");
  input1 = input.nextDouble();
  System.out.println("Enter first y coordinate: ");
  input2 = input.nextDouble();
  System.out.println("Enter second x coordinate: ");
  input3 = input.nextDouble();
  System.out.println("Enter second y coordinate: ");
  input4 = input.nextDouble();
  input.close();

  getSlope(input1, input2, input3, input4);
}
```

Printing the Slope

If we try to run the program, we will not see anything printed to the screen. Why? Because there is no print statement anywhere within the code. All the program does is calculate and return values. Returning values and printing them are **not** the same thing. Therefore, we need to include a print statement if we want to actually see the output. However, we cannot just include a print statement within our method because it is a `String` method, not a `void` one. Fortunately, our `main()` method is `void` so we can just ask the system to print the output after the method is called.

```java
static double input1;
static double input2;
static double input3;
static double input4;

/**
 * This method returns the slope between two sets
 * of coordinate points by calculating their coordinate
 * changes separately
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A String expression of the slope in rise / run format
 */
public static String getSlope(double x1, double y1,
                              double x2, double y2) {
  double yChange = y2 - y1;
  double xChange = x2 - x1;
  return yChange + " / " + xChange;
}

public static void main(String args[]) {
  Scanner input = new Scanner (System.in);
  System.out.println("Enter first x coordinate: ");
  input1 = input.nextDouble();
  System.out.println("Enter first y coordinate: ");
  input2 = input.nextDouble();
  System.out.println("Enter second x coordinate: ");
  input3 = input.nextDouble();
  System.out.println("Enter second y coordinate: ");
  input4 = input.nextDouble();
  input.close();

  System.out.println(getSlope(input1, input2, input3, input4));
} //prints what is returned by the getSlope() method
```

Lab: Helper Methods

Purpose of Helper Methods

When a method calls another method, it is using that method to help it perform a particular action. **Helper** methods provide users with more flexibility when it comes to developing methods. Additionally, helper methods help reduce code repetition because the same action only has to be written once. Let's start by creating a few helper methods that will help us with other methods later.

```
/**
 * This method returns the difference in y values
 *
 * @param y1 A double of the first y-coordinate
 * @param y2 A double of the second y-coordinate
 * @return The difference of y1 and y2 as a double
 */
public static double getRise(double y1, double y2) {
  return y2 - y1;
}

/**
 * This method returns the difference in x values
 *
 * @param x1 A double of the first x-coordinate
 * @param x2 A double of the second x-coordinate
 * @return The difference of x1 and x2 as a double
 */
public static double getRun(double x1, double x2) {
  return x2 - x1;
}
```

Above, we have two methods. One that calculates the *rise* of a slope and another that calculates the *run* of a slope. These two helper methods will come in handy in our later slope calculations.

Using Helper Methods

```java
/**
 * This method returns the slope in decimal form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A double expression of the slope
 */
public static double getSlopeDecimal(double x1, double y1,
                                      double x2, double y2) {
    return getRise(y1, y2) / getRun(x1, x2);
}

/**
 * This method returns the slop in fraction form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A String expression of the slope in rise / run format
 */
public static String getSlopeFraction(double x1, double y1,
                                       double x2, double y2) {
    return getRise(y1, y2) + " / " + getRun(x1, x2);
}
```

Notice how within the two methods above `getSlopeDecimal()` and `getSlopeFraction()`, the previous helper methods `getRise()` and `getRun()` are called. Having 4 methods at our disposal provides us with a flexibility that a single method cannot offer. In this program, we can get the slope in its decimal form and its fraction form in addition to the rise and run individually. If we wanted, we can continue to build more into this program.

Complete and Run the Program

Copy over the rest of the program and then test it.

```java
static double input1;
static double input2;
static double input3;
static double input4;
```

```java
/**
 * This method returns the difference in y values
 *
 * @param y1 A double of the first y-coordinate
 * @param y2 A double of the second y-coordinate
 * @return The difference of y1 and y2 as a double
 */
public static double getRise(double y1, double y2) {
  return y2 - y1;
}

/**
 * This method returns the difference in x values
 *
 * @param x1 A double of the first x-coordinate
 * @param x2 A double of the second x-coordinate
 * @return The difference of x1 and x2 as a double
 */
public static double getRun(double x1, double x2) {
  return x2 - x1;
}

/**
 * This method returns the slope in decimal form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A double expression of the slope
 */
public static double getSlopeDecimal(double x1, double y1,
                                      double x2, double y2) {
    return getRise(y1, y2) / getRun(x1, x2);
}

/**
 * This method returns the slop in fraction form
 *
 * @param x1 A double of the first x-coordinate
 * @param y1 A double of the first y-coordinate
 * @param x2 A double of the second x-coordinate
 * @param y2 A double of the second y-coordinate
 * @return A String expression of the slope in rise / run format
 */
public static String getSlopeFraction(double x1, double y1,
                                       double x2, double y2) {
    return getRise(y1, y2) + " / " + getRun(x1, x2);
```

```
}

public static void main(String args[]) {
  Scanner input = new Scanner (System.in);
  System.out.println("Enter first x coordinate: ");
  input1 = input.nextDouble();
  System.out.println("Enter first y coordinate: ");
  input2 = input.nextDouble();
  System.out.println("Enter second x coordinate: ");
  input3 = input.nextDouble();
  System.out.println("Enter second y coordinate: ");
  input4 = input.nextDouble();
  input.close();

  System.out.print("Rise: ");
  System.out.println(getRise(input2, input4));
  System.out.print("Run: ");
  System.out.println(getRun(input1, input3));
  System.out.print("Calculated form: ");
  System.out.println(getSlopeDecimal(input1, input2, input3,
        input4));
  System.out.print("Slope form: ");
  System.out.println(getSlopeFraction(input1, input2, input3,
        input4));
}
```

Lab Challenge: Greeting Machine

Create a Greeting Machine

You are going to develop a method that takes an ArrayList of strings as a parameter, iterates through that ArrayList and greets every element in it with `"Hello"` followed by a **newline**.

Existing Code:

```
import java.util.*;

public class LabChallenge {

  //add code below this line



  //add code above this line

  public static void main(String args[]) {
    ArrayList<String> names = new ArrayList<String>();
    for (String s : args) {
      names.add(s);
    }
    sayHello(names);
  }

}
```

▼ **Hint**

You can start your method with `public static void sayHello(ArrayList<String> variable)`. Then iterate through `variable` using a loop in order to print `Hello` to each element in `variable`. You can also review the reading question in **"Alternative Parameters"** in the **"Parameters"** assignment.

Requirements

- You **should not** make any changes to the code that already exists. If you accidentally delete any existing code, you can copy and paste the entire program from above.

- You can use **any** number of additional methods you'd like but you **must** include at least the method called `sayHello()` in your code.

Compile and test your code with a few different values

▼ **Expected Output**

```
Hello Alan
Hello Bob
```

▼ **Expected Output**

```
Hello 1
Hello 2
Hello 3
```

▼ **Expected Output**

```
Hello Codio
```