Code Activity: Writing Methods*

Instructions

Objectives

- Explore how write simple methods.

Background

In this exercise, you'll practice writing a simple **method** in your class. In all previous examples and classes, you have written code that exists in the main() method. As you know, when you execute a class, the starting point is that class's main() method. If you want any code to execute it will need to eventually lir back to some code that executed in main().

Most often, you'll write a method to compute a value based on given inputs. The computed value is said to be *returned* from that method. In this lab, we'll wo you through writing a method in your class and how to *call* it from the main() method.

A method (sometimes referred to as a function) has several parts:

- modifier
- return_type
- methodName
- parameters
- exceptions

Methods follow the following format:

modifier return_type methodName(parameters) exceptions {

//method body

Modifier

The modifier controls who (which classes) can access this method.

There are four modifiers and they are collectively known as **access modifiers** and they are **public**, **private**, **protected**, and the default access level. On the first 3 are reserved keywords in Java.

Return Type

The *return type* is a specified data type that your method will *return*. It can be any primitive type (boolean, int, char, etc.) or object type (a class). If your method doesn't return anything, you'll use the **void** keyword.

Method Naming Rules

When creating methods, they are a few rules that you should be aware of and conventions you should follow.

The following rules apply when writing a method:

- reserved keywords cannot be used. See the list in the Reference section.
- names cannot start with a digit.
- digits can be used after the first character (e.g., n11, n22e are valid method names).
- names can start with a letter, an underscore (i.e., "_") or a dollar sign (i.e., "$").
- names cannot use other symbols or spaces (e.g., "%","^","&","#").

Parameters

Parameters are input values that your method can accept. You can have any number of parameters that you like. A good rule of thumb though, is if your parameter list is long (above 6 items), then there is probably an easier way to write your method.

To declare a parameter you'll specify the datatype and a name for your parameter. The same rules that apply to variable names apply to parameter names.

Exceptions

Method Body

This is the main portion of your method that determines what it actually does. This is where you'll place lines of code. All methods should have an opening an closing curly brace to indicate the start and end of the method body. The body is just a line or several lines of code to execute whenever this method is callec executed.

Guided Practice

Now that you have some background on the basics of methods, let's write a few together.

Project Setup

Open your IDE (Eclipse), and select File > New > Java Project.

Provide the name, Lab-Method and click OK.

Right-click on the newly created project and select New > Class.

Provide the class the name, MethodDeclaration and click OK.

Now edit the file so that it looks like the following:

```
public class MethodDeclaration{

 public static void main(String[] args) {
  //create a class instance
  //call your first method here
 }
 //create your first method here
}
```

Next we'll create a method in our class. We'll provide the name *talk* for the method. It will be public, return nothing, and declare no parameters or exceptior

```
public class MethodDeclaration{

 public static void main(String[] args) {
  //create a class instance

  //call your first method here
 }

 //create your first method here
```

```
 }
}
```

Great! With the above code you have now created a method. It has a **void** return type (because it doesn't return anything), the method name is talk and it declares zero parameters.

The body of the method is a single statement that prints a message to the console.

One question still remains. How do we call our talk() method?

We'll first have to create an instance of a class and then call the method on it.

The next step we'll create an instance of our class. We'll place this inside of the main method:

```
public class MethodDeclaration{


 public static void main(String[] args) {
   //create a class instance
   MethodDeclaration md = new MethodDeclaration();


   //call your first method here
 }


 //create your first method here
 public void talk(){
   System.out.println("Inside of the talk method");
 }
}
```

Now the above addition is peculiar. We can create an instance of a class from within its own method? Yes. This is valid code.

Generally, only classes used for testing purposes or ones specialized in creation are used to create instances, but for brevity, we'll use this approach of creating an instance "within" a class.

Now that we have an instance to work with, we can *call* a method on this class.

Update your code to call the talk() method. In order to do so, you'll use the variable name that refers to the instance of MethodDeclaration that you created and then use the **dot operator** followed by the name of the method and any passing arguments.

Recall that the **dot operator** is used whenever you want to execute (call) a method on an object. You first specify a variable name, a period, and then the name of the method.

```
public class MethodDeclaration{


 public static void main(String[] args) {
   //create a class instance
```

```
   //call your first method here
   md.talk();
 }


 //create your first method here
```

public void talk(){

  System.out.println("Inside of the talk method");

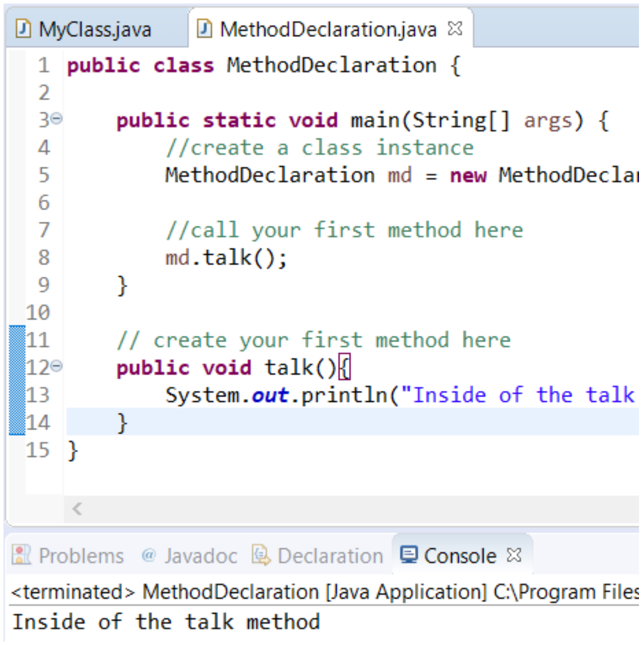
 }

}

Excellent. Here you've used the variable md (which refers to a single instance of MethodDeclaration) and called its talk() method. Notice that because talk i: method, we follow the name of it with a pair of parentheses. There is nothing inside of the parentheses because we aren't passing any arguments. This is because we didn't specify any parameters when we defined our method.

NOTE: *Arguments* are just values (or variables) that you provide to a method. When inside of a method, they are referred to as *parameters.*

Run the program. You'll see the output of the program.



```
public class MethodDeclaration {

    public static void main(String[] args) {
        //create a class instance
        MethodDeclaration md = new MethodDeclai

        //call your first method here
        md.talk();
    }

    // create your first method here
    public void talk(){
        System.out.println("Inside of the talk
    }
}
```

```
<terminated> MethodDeclaration [Java Application] C:\Program Files
Inside of the talk method
```

Method with an Argument

You can also create methods that accept arguments. All you need to do is specify a parameter for each input that you need.

Let's create a method that will accept one int type and return it back to the user.

Edit the file to add a new method.

public class MethodDeclaration{

 public static void main(String[] args) {

  //create a class instance

  MethodDeclaration md = new MethodDeclaration();

  //call your first method here

```
  md.talk();
 }

 //create your first method here
 public void talk(){
  System.out.println("Inside of the talk method");
 }

 public int getInt(int input){
  return input;
 }
}
```

The next thing we'll do is call the method in our main() method and specify an argument for it. We can either create the variable and pass it by name or direc specify the value.

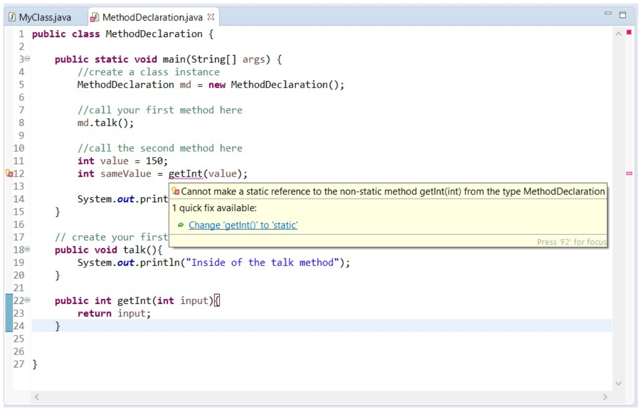
We'll specify the variable first and then pass it by name to the method.

```
public class MethodDeclaration{

 public static void main(String[] args) {
  //create a class instance
  MethodDeclaration md = new MethodDeclaration();

  //call your first method here
  md.talk();

  //call the second method here
  int value = 150;
  int sameValue = getInt(value);

  System.out.println(sameValue);
 }

 //create your first method here
```

```
 }

 public int getInt(int input){
  return input;
 }
}
```

Save the file.

Hmmm. Notice the compiler error?



```
public class MethodDeclaration {

    public static void main(String[] args) {
        //create a class instance
        MethodDeclaration md = new MethodDeclaration();

        //call your first method here
        md.talk();

        //call the second method here
        int value = 150;
        int sameValue = getInt(value);

        System.out.print
    }

    // create your first
    public void talk(){
        System.out.println("Inside of the talk method");
    }

    public int getInt(int input){
        return input;
    }


}
```

```
Cannot make a static reference to the non-static method getInt(int) from the type MethodDeclaration
1 quick fix available:
 Change 'getInt()' to 'static'
                                                              Press 'F2' for focus
```

If you mouse over the highlighted code a message will read, "Cannot make a static reference to a non-static method..."

We'll have a separate lesson on the meaning of **static**. If you find yourself running into this type of issue, it's because you forgot to specify the method as a p
of the object variable.You can only call methods by using the dot-operator on a variable that refers to the class type.

In this case, our variable *md* needs to call getInput(..).

Edit the file so that it calls getInt(..) from an instance of MethodDeclaration.

public class MethodDeclaration{

 public static void main(String[] args) {

  //create a class instance

  MethodDeclaration md = new MethodDeclaration();

  //call your first method here

  md.talk();

  //call the second method here

  int value = 150;

  int sameValue = md.getInt(value);

  System.out.println(sameValue);

 }

 public void talk(){

  System.out.println("Inside of the talk method");

 }

 public int getInt(int input){

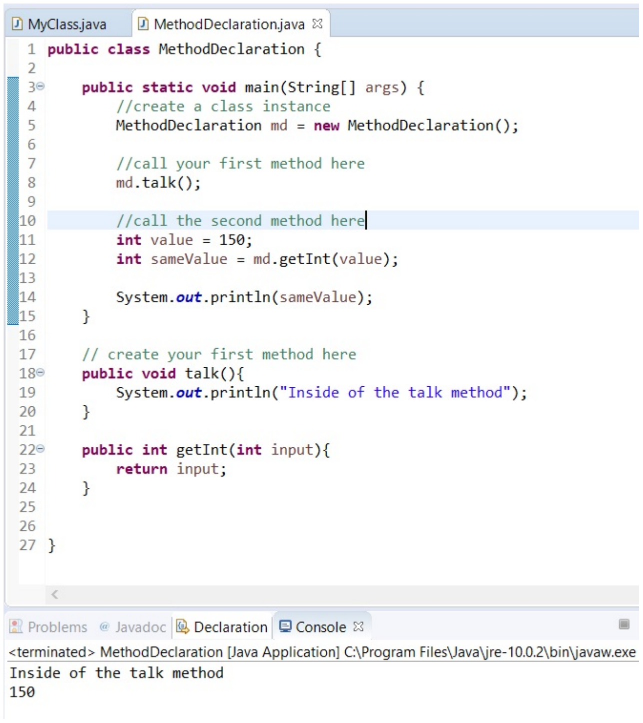
return input;

}

}

Run the program.

You'll see the output on the console of the value you passed to your method.



MyClass.java | MethodDeclaration.java

```java
public class MethodDeclaration {

    public static void main(String[] args) {
        //create a class instance
        MethodDeclaration md = new MethodDeclaration();

        //call your first method here
        md.talk();

        //call the second method here
        int value = 150;
        int sameValue = md.getInt(value);

        System.out.println(sameValue);
    }

    // create your first method here
    public void talk(){
        System.out.println("Inside of the talk method");
    }

    public int getInt(int input){
        return input;
    }


}
```

Problems | Javadoc | Declaration | Console

```
<terminated> MethodDeclaration [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe
Inside of the talk method
150
```

Multiple Parameters

We'll now create a method that accepts multiple parameters. All we need to do is separate each with a comma.

Edit the file to create a new method sum() that takes in three parameters. One should be a **double** and the other two can be an **int**:

public class MethodDeclaration{

public static void main(String[] args) {

//create a class instance

MethodDeclaration md = new MethodDeclaration();

md.talk();

//call the second method here

int value = 150;

int sameValue = md.getInt(value);

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

 //create your first method here
 public void talk(){
  System.out.println("Inside of the talk method");
 }

 public int getInt(int input){
  return input;
 }

 public double sum(int x, int y, double z){
  return x + y + z;
 }
}
```

Now call this method in the main() method and print the result to the console.

```
public class MethodDeclaration{
 public static void main(String[] args) {
  //create a class instance
  MethodDeclaration md = new MethodDeclaration();

  //call your first method here
  md.talk();

  //call the second method here
  int value = 150;
  int sameValue = md.getInt(value);

  System.out.println(sameValue);

  //call the third method here
```

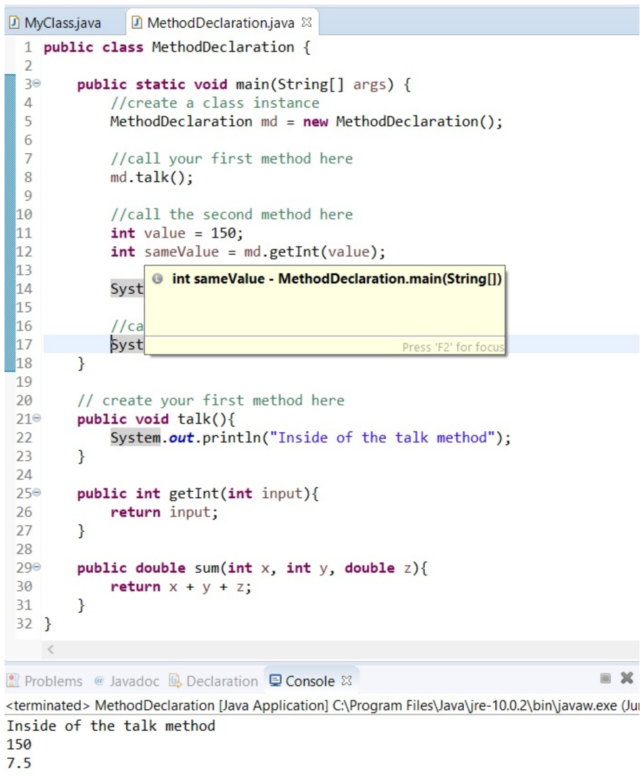
```
 //create your first method here
 public void talk(){
  System.out.println("Inside of the talk method");
 }
```

```
public int getInt(int input){
  return input;
 }

 public double sum(int x, int y, double z){
  return x + y + z;
 }
}
```

In the above code, we've called the sum method and specified its arguments all in one line.

Run the program.

You'll see the value of the result on the console.



```
public class MethodDeclaration {

    public static void main(String[] args) {
        //create a class instance
        MethodDeclaration md = new MethodDeclaration();

        //call your first method here
        md.talk();

        //call the second method here
        int value = 150;
        int sameValue = md.getInt(value);

        Syst   int sameValue - MethodDeclaration.main(String[])

        //ca
        Syst                                  Press 'F2' for focus
    }

    // create your first method here
    public void talk(){
        System.out.println("Inside of the talk method");
    }

    public int getInt(int input){
        return input;
    }

    public double sum(int x, int y, double z){
        return x + y + z;
    }
}
```

```
<terminated> MethodDeclaration [Java Application] C:\Program Files\Java\jre-10.0.2\bin\javaw.exe (Ju
Inside of the talk method
150
7.5
```

Congratulations. You've created 3 different methods.

Upload your work in the provided form.

Work on Your Own

Now that we've walked you through an exercise, it's time for you to try one on your own. You can check your work at the end with our posted solution in the Reference section.

For this exercise, we want you to create a methods that should return a random number between 1 and 50, inclusively. This method should be called randomNumber(), it should be have a public access modifier and return an int datatype. It should not declare any parameters or exceptions.

You'll need to use the Math class to generate a random number. It has a method called **random** that will return a value greater than or equal to 0.0 and les than 1.0. An example of its usage is below:

double d = Math.random();

Feel free to check your solutions against ours located in the Reference section.

When you are done, please Mark this activity Complete.

Description

Explore how to write simple methods

REFERENCE DOCUMENTS AND ADDITIONAL NC