STEM FUSE
A Revolution in STEM Education

Mobile App:IT

Methods & Classes

WHAT IS A METHOD?

- A **method** is a set of code which is referred to by name and can be called (**invoked**) at any point in a program simply by utilizing the method's name.

- This makes it easy to reuse portions of code so they do not have to be rewritten.

WHAT IS WRONG WITH THIS CODE?

Here is some code that greets people and says goodbye as they enter and leave the classroom.

```
System.out.println("Hello John, welcome to our class today.");
System.out.println("Hello Adam, welcome to our class today.");
System.out.println("Hello Tina, welcome to our class today.");
System.out.println("Hello Jane, welcome to our class today.");
System.out.println("Hello Alex, welcome to our class today.");
System.out.println("Goodbye John, thanks for coming to class.");
System.out.println("Hello Carter, welcome to our class today.");
System.out.println("Goodbye Adam, thanks for coming to class.");
```

THE ANSWER

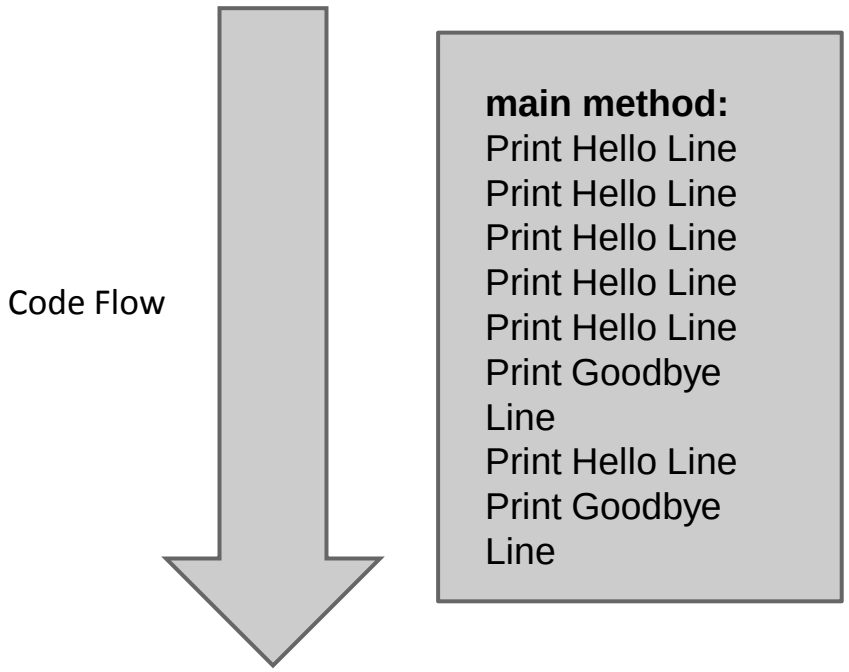
- While there is nothing wrong with the previous code, you will notice that you are having to rewrite large portions of the code over and over again.

- Imagine if you had 20 or more people coming in and leaving a class. That is a lot of extra typing!

- This is where a **method** can come in handy.

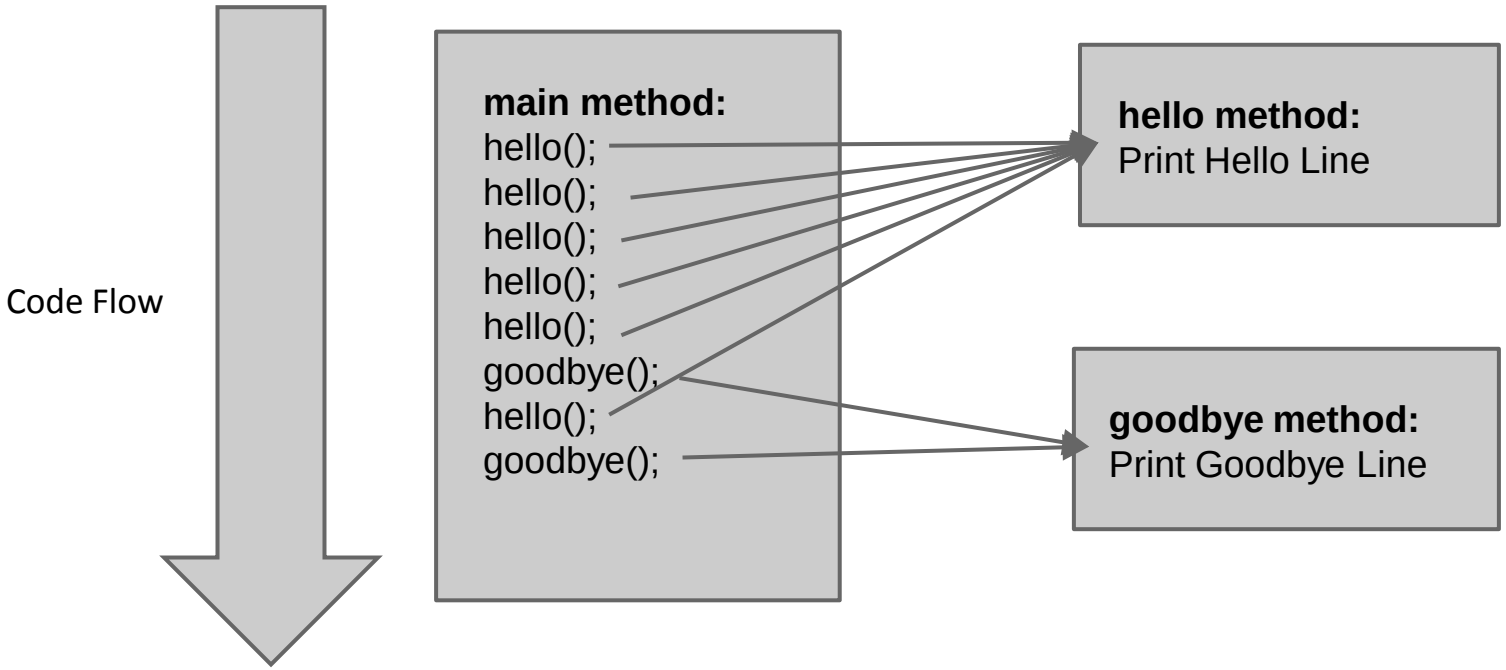
CURRENT PROGRAM INSTRUCTIONS

Here is how the program is currently working:



METHOD PROGRAM INSTRUCTIONS

Here is how the program can work if we break it into methods. Notice the difference?



YOU ALREADY KNOW METHODS

- You have previously used the println and print methods in your previous programs. This means you are already familiar with how method calls work. When you want to print a line of text, you make a call to one of the print methods.

- You do not need to know how the method works, you just need to know what the method does and how to call it.

DIFFERENT METHOD STYLES

There are two different types of methods:

- **Built In** - These are methods that are part of the Java compiler. An example of this type of method is System.out.println

- **User Defined** - These are methods that you can create and use in your own programs. You control the name of these methods and what they do.

WRITING OUR FIRST METHOD

Here is an example of how a basic method is written. Here we are building a simple hello method.

```
public static void hello() {
    System.out.println("Hello John, welcome to our class today.");
}
```

Explanation:

- "public" tells the program that anyone can use this method. The word "public" is optional in our example.
- "static" tells the program that this method does not require an **instance** of an object to be called.
- "void" tells the program that this method is not going to return anything. We will talk more about returning values later.

EXPLANATION CONTINUED

```
public static void hello() {
    System.out.println("Hello John, welcome to our class today.");
}
```

- "hello" is the name we want to give our method. This could be anything we want.
- () is used to let the program know that we are not going to pass any variables (called **parameters**) into this method. More on method **parameters** later.
- The method is then enclosed in {} (often called curly brackets). Anything inside these brackets gets run every time the method is called.

CODE STRUCTURE

- You will notice that within the hello method, the println line is indented.

- This indentation does not control how the program functions, however it makes the program much easier to read.

- The indentation is simply a tab character.

- When building your programs it is important to focus on keeping your programs easy to read and keeping the indentation correct.

FOLLOW ALONG WITH ECLIPSE

- Now it is time to follow along with the code samples in the upcoming slides.
- The first step is to open Eclipse and create a new project called "Greetings".
- After creating the project, Create a new Java class called "Greetings". Make sure to check the box to add the "public static void main" method.
- At the end of each code section, there will be a slide showing the full code. The code on these end slides will help to validate that you are putting the code in the correct location and you are maintaining the correct indentation structure.

ADD THE FIRST METHOD

The first step is to add the first method we built in a previous slide. This should be placed outside of the main method but within the Greetings class.

```
public static void hello() {
    System.out.println("Hello John, welcome to our class today.");
}
```

FIRST METHOD CALL

Now we will add a call to the hello method. Inside the main method add a call to hello.

```
/**
 * @param args
 */
public static void main(String[] args) {
  hello();
}
```

RUN YOUR PROGRAM

Your program should look like this, go ahead and run it now. You should have one line printed out.

```
public class Greetings {

    /**
     * @param args
     */
    public static void main(String[] args) {
      hello();
    }

    public static void hello() {
        System.out.println("Hello John, welcome to our class today.");
    }

}
```

CALLING THE METHOD MULTIPLE TIMES

You can now reuse the hello method.

```
/**
 * @param args
 */
public static void main(String[] args) {
  hello();
  hello();
  hello();
}
```

The only problem with this is that it just prints the same message three times.

METHOD PARAMETERS

- Wouldn't it be great if we could get the name in the message to change?

- With **method parameters** and **variables** it is possible.

- Parameters are the variables that are listed as part of a method declaration. Each parameter must have a unique name and a defined data type.

METHOD PARAMETERS

- The first step is to determine what variable type (**data type**) we want our method parameter to use. In our example we will use String.

- The second step is to decide on a **variable name**. In our example we will use first_name.

- The third step is to add the parameter to the method definition. Lets do that now.

ADDING THE PARAMETERS

```
public static void hello(String first_name) {
    System.out.println("Hello John, welcome to our class today.");
}
```

You will notice that after you change your hello method, your hello method calls look like this.

```
/**
 * @param args
 */
public static void main(String[] args) {
  hello();
  hello();
  hello();
}
```

ERRORS AND DEBUGGING

- These pink squiggly lines let you know that there is an error in your program.

- If you hover over the pink lines you will see an error message that can help you **debug** the error.

- The definition of **debug** is to identify and remove errors from (computer hardware or software).

RUNNING A PROGRAM WITH ERRORS

- If you try to run your program with these errors in place, you will receive an error message.

- If you proceed with running the program, there will be error messages listed in the Console window.

- Now we will fix these errors.

CALLING A METHOD WITH A PARAMETER

To fix the errors, we simply need to pass in a String variable to our hello method.

```
/**
 * @param args
 */
public static void main(String[] args) {
  hello("John");
}
```

Go ahead and run your program now. You will notice that it runs without errors.

MULTIPLE METHOD CALLS

Now we will go ahead and make a few more hello method calls with different first names.

```
/**
 * @param args
 */
public static void main(String[] args) {
  hello("John");
  hello("Adam");
  hello("Tina");
}
```

If you run the program, you will notice that there seems to be a problem. Do you see what the problem is?

USING THE FIRST_NAME VARIABLE

We are not using the first_name variable that is getting passed into our hello method. We already know how to use variables, so let's go ahead and do that now.

```
public static void hello(String first_name) {
    System.out.println("Hello" + first_name + ", welcome to our class today.");
}
```

Now run the program. The output should look much better now, but there is still a spacing error. You can fix this one on your own before you move on.

```
HelloJohn, welcome to our class today.
HelloAdam, welcome to our class today.
HelloTina, welcome to our class today.
```

MULTIPLE PARAMETERS

- Now let's add a second parameter to the hello method.
- Multiple parameters can be added and separated by commas.

Try This:

- Make a 2nd parameter for the hello method.
- Make it a string **data type.**
- Give it a **variable name** of "last_name".
- Add "last_name" to the print statement inside the hello method.
- Add a last name to each of the hello method calls inside the main method. Use the last names "Smith", "Jones", and "Johnson"

PRINTING THE FULL NAME CODE

```
public class Greetings {

    /**
     * @param args
     */
    public static void main(String[] args) {
      hello("John", "Smith");
      hello("Adam", "Jones");
      hello("Tina", "Johnson");
    }

    public static void hello(String first_name, String last_name) {
        System.out.println("Hello " + first_name + " " + last_name + ", welcome to our class today.");
    }

}
```

MORE PARAMETERS

- The next step is to add the time the student checks into the class.
- We will do this by adding two more parameters to the hello method.

Try This:

- Add two more parameters, both integer **data types** to the hello method.
- Name the two new variables "hour" and "minute"
- Add the times to the hello method calls.

Name	John	Adam	Tina
Hour	9	9	9
Minute	25	27	32

MORE PARAMETERS (TRY THIS)

- Break up the print statements to use multiple print method calls instead of just one println method call.

- Set it up so the output will display:

"Hello John Smith, welcome to our class today. The time is 9:25."

This one is a little more difficult so do not get discouraged. Try to walk through each step one at a time and use the past examples for help.

THE SOLUTION

```
public class Greetings {

    /**
     * @param args
     */
    public static void main(String[] args) {
      hello("John", "Smith", 9, 25);
      hello("Adam", "Jones", 9, 27);
      hello("Tina", "Johnson", 9, 32);
    }

    public static void hello(String first_name, String last_name, int hour, int minute) {
        System.out.print("Hello ");
        System.out.print(first_name);
        System.out.print(" ");
        System.out.print(last_name);
        System.out.print(", welcome to our class today. The time is ");
        System.out.print(hour);
        System.out.print(":");
        System.out.print(minute);
        System.out.println(".");
    }

}
```

THE OUTPUT

```
Hello John Smith, welcome to our class today. The time is 9:25.
Hello Adam Jones, welcome to our class today. The time is 9:27.
Hello Tina Johnson, welcome to our class today. The time is 9:32.
```

We will continue adding even more to the Greeting class so it is important to make sure you are able to get the same output that is listed above.