Module 1-5

Inputs and Outputs

- Methods
- Command Line

Methods

Methods

- Methods are **related** (hint: {...}) statements that complete a specific task or set of tasks.
- Methods can be called from different places in the code.
- When called, inputs can be provided to a method.
- Methods can also return a value to its caller.

Methods: General Syntax

Here is the general syntax:

```
access Modifier   return type   name of the method (... arguments…) {

    // method code.

}
```

- The return type can be one of the data types (boolean, int, float, etc.) we have seen so far.
- If the return type is “void” it means nothing is returned by the method.

Methods: Example

Here is an example:

```
public class MyClass {

        public int addTwoNumbers(int a, int b) {
                return a+b;
        }
}
```

The method addTwoNumbers is a method of the MyClass class.

The method expects 2 parameters as input. More specifically, it expects 2 integers

The method has a return value of int, so there needs to be a return statement that returns an integer.

Methods: Example

Here is a specific example of a void method:

```
public class MyClass {

	public void addTwoNumbers(int a, int b) {
		System.out.println(a+b);
	}
}
```

This method is void, thus has no return statement.

Methods: Calling A Method

Methods can be called from other methods.

```
public class MyClass {
        public int addTwoNumbers(int a, int b) {
                return a+b;
        }

        public String printFullName(String first, String last) {
                return last + ", " + first;
        }

        public void callingFunction (String args[]) {

                int result = addTwoNumbers(3,4);
                System.out.println(result);
                // result will be equal to 7.

                String fullName = printFullName("Andy", "Chong");
                System.out.println(fullName);
                // result will be equal to "Chong, Andy"
        }}
```

In here, we call the method **printFullName** from **callingFunction**, providing all needed parameters and saving the result intoresult.

Methods: Calling A Method

Once a method has been defined, it can be called from

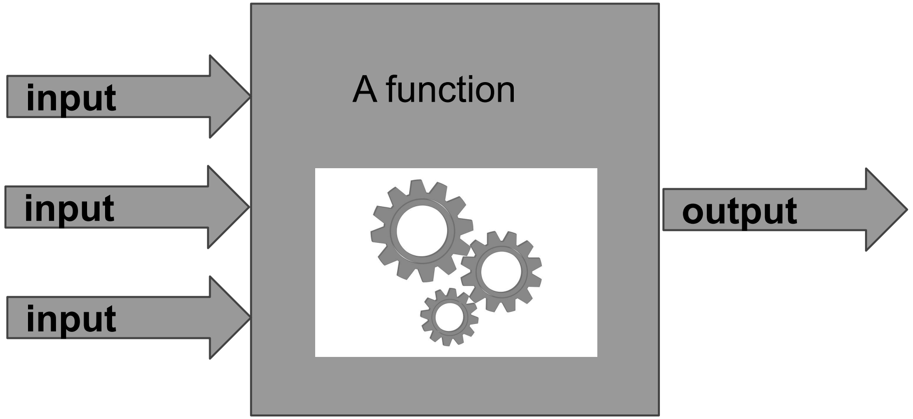
```
public class MyClass {
        public int addTwoNumbers(int a, int b) {
                return a+b;
        }

        public void callingFunction (String args[]) {

                int result = addTwoNumbers(3,4);
                System.out.println(result);
                // result will be equal to 7.
        }
}
```

addTwoNumbers takes 2 inputs, an integer a and an integer b. These are known as **parameters**.

When we call **addTwoNumbers**, we must provide the exact inputs specified (in this case 2 integers).

Methods: Factory Analogy

Methods are Java’s versions of functions. You can think of this as a process that could potentially take several inputs and use it to generate output.



Command Line Input / Output

Getting Input from the Command Line

- All programming languages must have the ability to read in data (input)
- Examples of input: a file, data being transmitted from a network, or **data typed in by the user**.

Using the Scanner Object

```
import java.util.Scanner;

public class InputReader {

        public static void main(String[] args) {

                Scanner userInput = new Scanner(System.in);

                System.out.print("Please enter your name: ");
                String name = userInput.nextLine();

                System.out.print("Please enter your height: ");
                String heightInput = userInput.nextLine();
                int height = Integer.parseInt(heightInput);

                System.out.println("Your name is: " + name + ".");
                System.out.println("Your height is:  " + height + " cm's.");
        }
}
```

To use the scanner object, we must import in the correct class.

Create an object of type Scanner

The input is read and stored into two Strings.

heightInput is converted into an int using the **Integer Wrapper Class.**

Reading In Multiple Items

This is one possible way to handle input for more than one item.

```
Scanner userInput = new Scanner(System.in);
System.out.print("Please enter several objects: ");
String lineInput = userInput.nextLine();

String [ ] inputArray = lineInput.split(" ");

for (int i=0; i < inputArray.length; i++) {
        System.out.println(inputArray[i]);
}
```

- When prompted a user enters each item separated by a space.
- The split method separates out each time using the spaces, and puts all of the items into an array!

Wrapper Classes

- Up until now, we have seen most of the **primitive** data types, to name a few: **int**, **boolean**, **char**, **long**, **float…**
- You have seen one r**eference** type: **Arrays**
- You might have noticed that non-primitive types seem to have extra functionality that can be invoked with the dot operator, for example: (**myArray.length**).
- All the primitive data types have more powerful non-primitive equivalents, these are called **wrapper classes**. You have seen an example of this.

Wrapper Classes Examples

- We have the following primitive data types: boolean, byte, char, float, int, long, short, double.

- These are the wrapper equivalents: Boolean, Byte, Character, Float, Integer, Long, Short, Double.

Wrapper Usage Examples

Wrapper classes come equipped with the helpful toString() method, which allows us to transform the data into a String. Consider this example:

```
Integer i = 9;
String iStr = i.toString();
```