CS620c

Methods writing, and using them correctly.

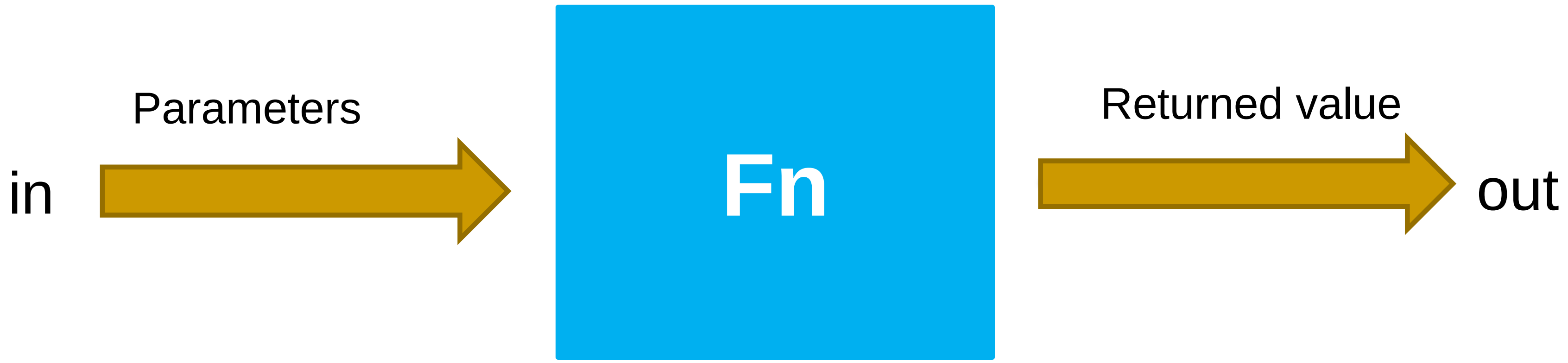
Callan Building Room 2.108

Email: Joseph.Duffin@nuim.ie

What are methods?

- A java method is a bundle (or packet) of lines of code with a common **function (Fn)**.
- When defined (declared) in a program it can be used (called or invoked) repeatedly.
- Methods can be called in loops and the loop will operate as before.
- Methods also describe the behaviour of Classes in object oriented programming (more to follow on this)

Method as a function (Fn)



Writing custom made method

• Writing methods helps to keep program code manageable.

•It allows us to maintain important code or algorithms that are used repeatedly in different places in programs.

•It allows the sharing of functionality with other parts of our software system (more to follow on this).

Methods encountered to date

•You have encountered a number of different methods in your code.

•You have written a main method in your programs and you have used the methods below. (You have called these methods, you did not have to declare them as they were already declared. Therefore you used them as prewritten tools)

```
System.out.println("Hello world");

myScanner.nextLine();// method call on the object myScanner

myString.charAt(4); // method call on the string object myString
```

Key Features of Methods

• A method <u>must</u> be written or declared before it can be used. (The only "home made" method you have used so far is the main method all the other methods that you used were premade for you)

•After a method is declared it can be used by calling or invoking it in your program.

The line of code below is described as a call to the charAt method and as you can see this method is used by providing it with a int parameter (actual parameter).

4 is the actual parameter supplied to the charAt method when it is used (called or invoked) in your program.

```
      myString.charAt(4);
```

Further rules for declaring and using methods

- The method declaration tells the programmer how the method should be used. (In fact the first line of the **method declaration** tells the programmer how the method should be used. This first line is the **method signature**)

- Overall the declaration is the design of the method and the method must be used (called or invoked) according to its design.

- For example if a method is designed to take one parameter as in the formal parameter list then writing more than one parameter or none when **calling the method** will generate and error.

- The **number of parameters** in the call to the method (**actual parameter list**) must match the type and number of parameters in the method signature (**formal parameter list**) of the method declaration.

- If a method signature has a return type other than **void**, then the method statements must contain the **return** statement.

- Method names have to obey the rules for **legal identifiers** in Java but as a code convention rule the first letter in the identifier is a small letter, thereafter the remaining words in the identifier have a their first letter capitalised. E.g **addTwoNumbers( ).**

In brief: rules for writing and using methods

1. The use or call of a method must match the signature declaration of the method.

2. If the method returns a value of a particular type then the receiving variable should be capable of storing that type.

3. The number and order of the parameters in the method call (actual parameter list) must match the number and order of parameters in the method signature declaration (formal parameter list).

4. The type of the parameters in the both the method call and the method declaration must match.

General structure of a method in Java

The Method Declaration (definition) is made up of (a) 1st line of the declaration called the method signature and (b) the code between the braces "{ }" is called the method body

```
access_modifiers  return_type  method_name (formal parameter list)
{
   statements; // if the return type is not void then you need a return statement here.
}
```

Note: The formal parameter list can be empty or made up of comma separated declared parameters as in the example below, int a, int b**.**

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

The method above is used in your code by typing:

```
       int  x = addTwoNumbers(10,27); // method call
```

10 and 27 are the actual parameters and these are copied into the formal parameters. The variable **x** eventually will receive the returned value from the method.

What happens when we make a call addTwoNumbers?

1 A memory resource called a **stack frame or run time stack** is created for the call to the method addTwoNumbers(10,27).

2 Memory space is set aside within this **stack** for the formal parameters **int a** and **int b (and local variables if the are declared)**

3 The **return point or address** is stored in the **stack**. This is the point at which the program continues to execute after the method is finished running.

4 The values in the actual parameter s (10, 27) are **<u>copied</u>** into the formal parameters **int a** and **int b**. 10 is copied into **a** and 27 is copied into **b.**

5 The code statements which were written in the body of the method are now executed in this case a is added to b.

6 The value of **a** added to **b** is returned to the point from which the method was originally called using the **return address** to find out where to go**.**

7 The **variable x** takes on the returned value which is 37.

8 Finally the stack frame is “destroyed” or returned to the system to be reused. All transient information in the stack frame is therefore destroyed. The Formal parameters **int a** and **int b** are no longer available in your program.

Your program continues executing beyond the method call onto the next line of code.

NB: The stack frame or run time stack will stay “alive” as long as the method has code still remaining to be executed or until a return statement is reached in the method body.

The main method (a familiar friend)

The main method is the entry to running most programs. It can have the method signature below and it has a special status with the **C** language from which Java was developed.

```
public static void main()
```

– It is special because when you hit the "run" button on many IDEs, this will run the main method (or run the code inside the { } of the main method).

String [ ] args in the main method

One thing we note is we can omit the String[] args part of this method declaration.

```
public static void main ()
```

- This does not make any difference to running our code within the main method but we will see later in this module how we can provide information to our main method when we run it.

Writing your own: a simple method to take single value

```
public static int timesTwo(int inValue)
```

- Now we are going to write a very simple method called timesTwo.

- timesTwo will now become a command within Java.

- It accepts a single integer parameter, which we will name `inValue`.

- This method will calculate two-times the value in the variable `inValue` and <u>return this value</u>.

Parameters

Parameters

- Formal parameters are special variables.
- They are use to pass information into a method.
- They act like ordinary variables

– but only inside the method! (remember the runtime stack and what happens when a method is called!)

```
public class MyClass{

     public static void timesTwo(int inVal){
          System.out.println(inVal * 2);
     }
}
```

Parameters

Parameters

• We regularly pass actual parameters into standard Methods.

– System.out.println(“This is a String parameter”);

– myString.charAt(2);

Invoking (calling) methods

Invoking Methods

• The main() method (and other methods) can now "call" or "invoke" the timesTwo method

• The caller will pass a value into the method through its parameter(s)

• The parameter like a variable within the method

Parameter passing to a method

Parameters

```
public class MyClass{

   public static void main () {
          timesTwo(3);
          timesTwo(99);
     }

   public static void timesTwo(int inVal){
          System.out.println(inVal * 2);
     }

}
```

It does not matter if the main method is before or after your declared methods

Parameters

```
public class MyClass{

     public static void main () { //main first
          timesTwo(3);
          timesTwo(99);
     }

     public static void timesTwo(int inVal){
          System.out.println(inVal * 2);
     }
}
```

Parameter passing to a method

Parameters

```
public class MyClass{

        public static void timesTwo(int inVal){

                System.out.println(inVal * 2);
        }
        public static void main () {
                timesTwo(3);  //first method call
                timesTwo(99); //second method call
        }
}
```

- Executing main() above, will generate the following output

```
6
198
```

Two kinds of methods, returning or not returning

Remember these two standard methods?

```
1.  System.out.println(“This is a String parameter”);

2. myString.charAt(2);
```

- We can identify two types of static method
- 1. just did its job and no more
- 2. did its job, but then returned a value at the end

No return value (void return type)

No return value (void)

• We could say that the `.println()` method did not return a value
• Methods that do not return a value are defined with a return type of void.

– It sends data to the output device, but that's different

• You could never say
• `x = System.out.println();`

• Because `println()` isn't a value

Returning a value.

A return value

- However `.charAt()` method does return something a character value
- `.charAt()` has a Return type of char

`String myString = "abcd";`

– `myString.charAt(2)` returns `'c'`

- You can say

– `char charVariable = myString.charAt(2);`

Returning value, the twoTimes method.

Return values – twoTimes

- I would like a new version of twoTimes method to return the doubled value
– rather than just print it to the screen

- We change the type of the method from void to int.

```
public class MyClass{

      public static int twoTimes(int inVal){
            return inVal * 2;
      }
}
```

Returned value (from calling the method)

- My new twoTimes method returns a value.
- That means that the line calling twoTimes must expect a returned value to arrive from twoTimes.

```
public class MyClass{

     public static int twoTimes(int inVal){
          return inVal * 2;
     }
     public static void main () {
          int result = twoTimes(3);
     }
}
```

Using the returned value

Use returned values

• It would be illegal Not to do something with the returned value.

– Use it, Store it... but don’t ignore it!

```
public static int twoTimes(int inVal){
        System.out.println(inVAl * 2);
}

public static void main () {

        int result = twoTimes(3);
        System.out.println(twoTimes(99));
}
```

Multiple Parameters

Multiple Parameters

• Some methods need more than 1 parameter

• We separate individual parameters with commas

• Suppose we are writing a big program that frequently calculates the average of two numbers. It might be useful to be able to call

```
average(firstScore, secondScore)
```

Multiple Parameters.

Multiple Parameters

- Many parameters can be listed in the method definition, separated by commas (,)
- Lets assume they are all integers

```
public static int average(int a, int b)
{
      return (a + b)/2;
}
```

Conclusion

• Methods are custom made “commands” or functions that can be used by Java programs as well as in many other languages

• Some methods are designed to take 1 or more parameters while are designed to accept **no** parameters:

Parameters can be primitive types; int, double, char etc or they can be (link references to the following) objects like Strings or objects that you define in your program, or arrays of primitive types or arrays of objects.

• Some methods are designed to return one result value while others are designed to **not** return a value (in this case the return type is **void** in the method signature).

Do you know the page in last week’s notes titled “What happens when a method is called?” ?