CS620c

Introducing scope of variables, using your public static int sigma function in another program.

Joe Duffin

Callan Building Room 2.108

Email: Joseph.Duffin@nuim.ie

What is scope? And what are the scope rules?

Every variable in Java is characterised by a duration and a scope. A variable's duration is the time during which it exists, and **its scope is the portion of the program from which the variable can be addressed.**

There are two types of scope, (1) block scope (2) class scope. In this lesson we will focus on block scope but we will be returning to class scope later in the module and we will refer to this concept of scope throughout the module.

The **scope of a variable** is the part of the program in which you can access it. The scope of a local variable ranges from its declaration until the end of the **block** or **for** statement in which it is declared.

The scope of a method's formal parameter variable is the entire method.

A variable that is defined in within a method is called a local variable. The scope of a local variable ranges from its declaration until the end of the **block { }** or for statement (**for (int I = 1;.....**) in which it is declared.

The variable declared in a **for** statement only extends to the structure of that statement.

Scope example; for loop statement

The scope of a local variable (**int square**) ranges from its declaration until the end of the block in which it is declared.

```
public static void main(String []args){

     int sum = 0;
     for (int i=1; i<=10; i++){
          int square = i*i;
          sum = sum + square;
     }
     // variable square is not available here
     System.out.println(sum);

} // end of main
```

Scope example; for loop statement

The variable declared in a **for** statement (**int i**) only extends to the end of the **for** statement. After the **for** statement finishes, the variable is no longer available or accessible. (its scope is the **for loop block** in which it was **declared**).

```
public static void main(String []args){
     int sum = 0;
     for (int i=1; i<=10; i++){
          sum = sum + i*i;
     }
     // variable i is not available here
     System.out.println(sum);
} // end of main
```

Scope example, overlapping scope error.

It is **illegal** to declare two variables with the same name in the same method in such a way that their scopes overlap. The code below is **illegal** because the method scope overlaps with the for loop scope. The **variable n** is declared twice in overlapping scopes.

```
public static int sumOfSquares (int n){

     int sum = 0;
     for (int i=1; i <= n; i++){

               int n = i*i; //ERROR
               sum = sum + n;
          }

     return sum;
}
```

Scope example of non-compiling code

This code will not compile. The scope of main does not extend outside main. The variable **sideLength** is not accessible in the scope for cubeVolume. (**BlueJ** has colour coded scope to remind you!)

```
public static void main(String []args){

     double sideLength = 10; // Error
     int result = cubeVolume();
     System.out.println(result);
}
```

```
public static int cubeVolume(){

     return sideLength*sideLength*sideLength;
}
```

Scope example; Everything is fine here!

This code will compile because the variable **int result** is declared in two separate methods whose scopes do **not** overlap.

```
public static void main(String []args){

     int result = square(3) + square(4); //OK
     System.out.println(result);
}
```

```
public static int square(int n){

     int result = n * n; // OK
     return result;
}
```

Scope example; Two variables with the same name!

This code will compile because the variable **int i** is declared in two different **for loop** scopes, even within the same method.

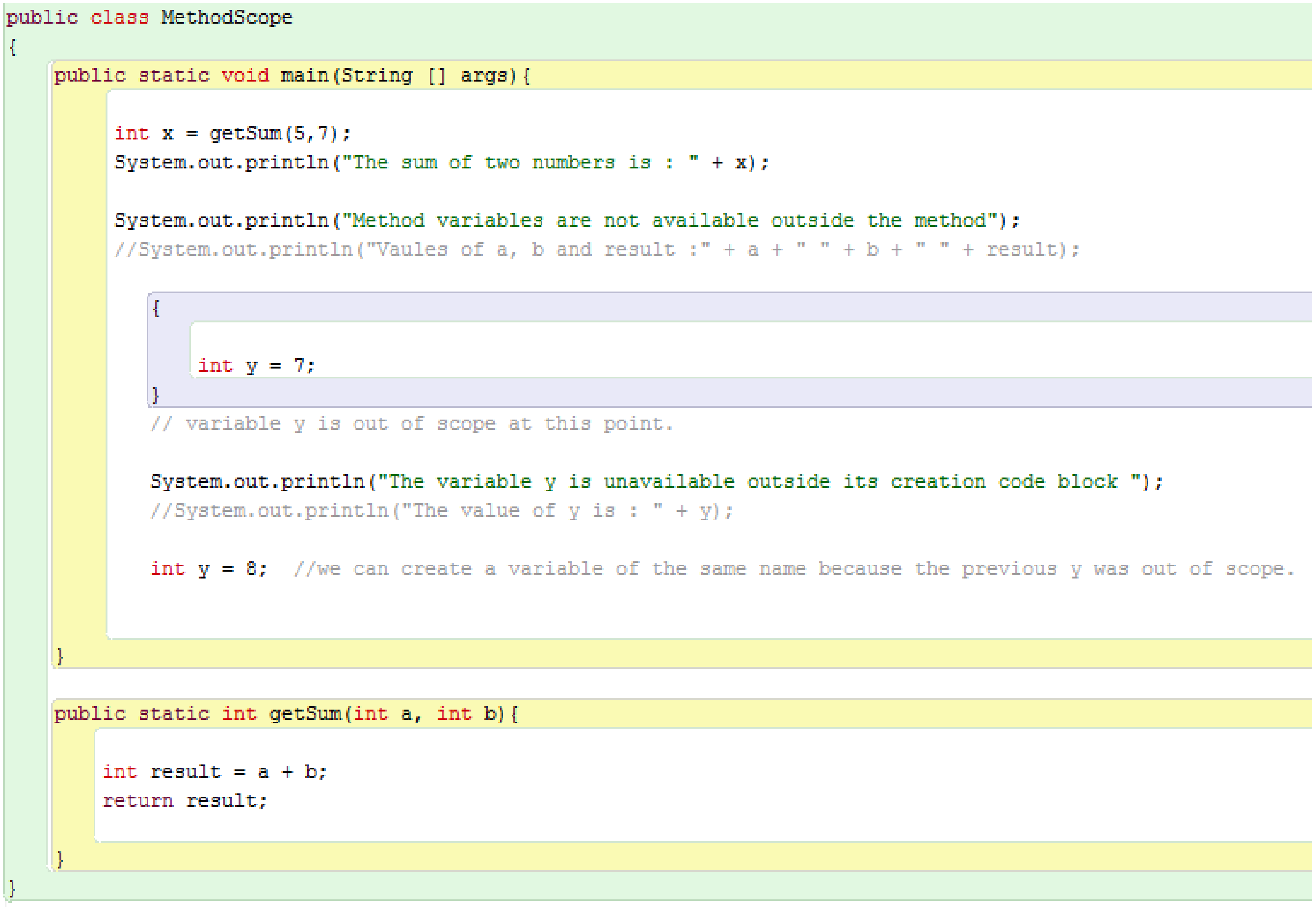
```
public static void main(String []args){
      int sum = 0;
      for (int i = 1; i <= 10; i++){ //OK
            sum = sum + i;
      }

      for (int i = 1; i <= 10; i++){ //OK
            sum = sum + i * i;
      }

      Sytem.out.println(sum);
}
```

Overlapping scope example; Two variables with the same name!

This code will **not** compile because the variable **int i** is declared in a scope that spans the whole method { }. Therefore you can not declare i again in a scope within this scope {}.

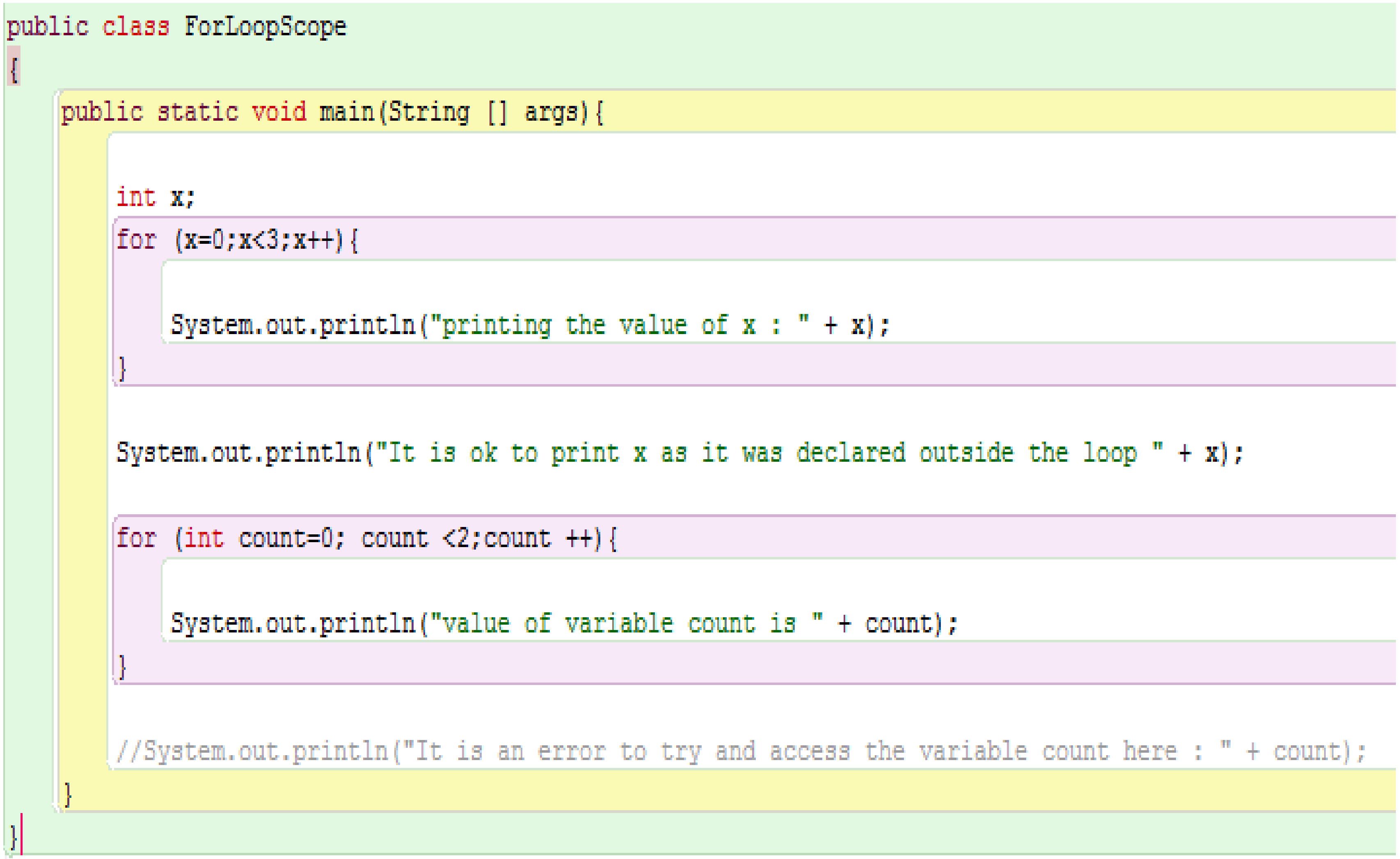
```
public static void main(String []args){
    int sum = 0;
    int i = 0;
    for (i = 1; i <= 10; i++){ //OK
              sum = sum + i;
    }

    for (int i = 1; i <= 10; i++){ //Error
              sum = sum + i * i;
    }

    Sytem.out.println(sum);
}
```

Scope examples; (1)code block; (2) method call



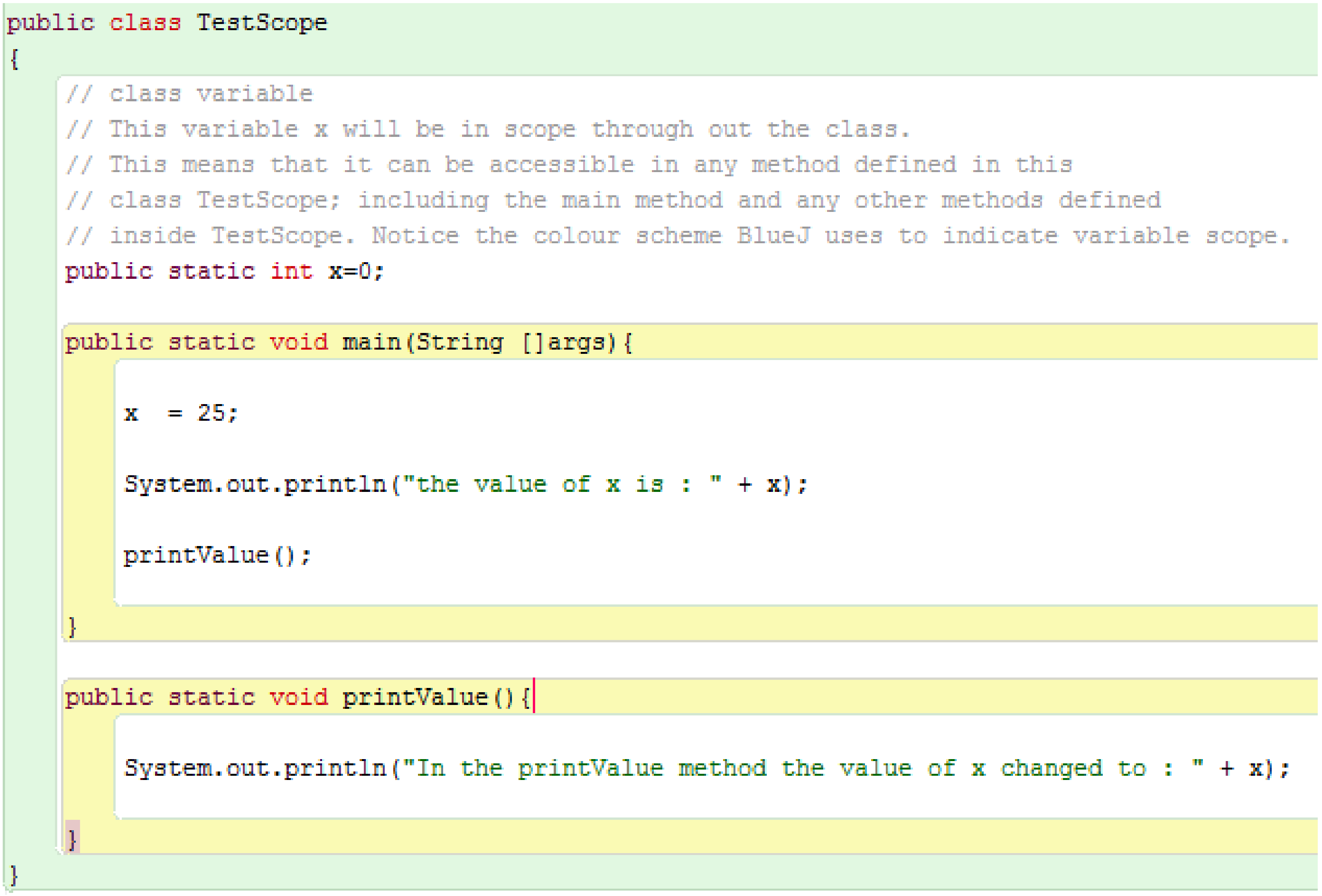
```java
public class MethodScope
{
    public static void main(String [] args){

        int x = getSum(5,7);
        System.out.println("The sum of two numbers is : " + x);

        System.out.println("Method variables are not available outside the method");
        //System.out.println("Vaules of a, b and result :" + a + " " + b + " " + result);

           {

               int y = 7;
           }
           // variable y is out of scope at this point.

           System.out.println("The variable y is unavailable outside its creation code block ");
           //System.out.println("The value of y is : " + y);

           int y = 8;  //we can create a variable of the same name because the previous y was out of scope.

    }

    public static int getSum(int a, int b){

        int result = a + b;
        return result;

    }
}
```

Scope examples; inside and outside for loop



```
public class ForLoopScope
{
    public static void main(String [] args){

        int x;
        for (x=0;x<3;x++){

            System.out.println("printing the value of x : " + x);
        }

        System.out.println("It is ok to print x as it was declared outside the loop " + x);

        for (int count=0; count <2;count ++){

            System.out.println("value of variable count is " + count);
        }

        //System.out.println("It is an error to try and access the variable count here : " + count);
    }
}
```

Class Scope of a class variable



```
public class TestScope
{
    // class variable
    // This variable x will be in scope through out the class.
    // This means that it can be accessible in any method defined in this
    // class TestScope; including the main method and any other methods defined
    // inside TestScope. Notice the colour scheme BlueJ uses to indicate variable scope.
    public static int x=0;

    public static void main(String []args){

        x  = 25;

        System.out.println("the value of x is : " + x);

        printValue();

    }

    public static void printValue(){

        System.out.println("In the printValue method the value of x changed to : " + x);

    }
}
```

A Sigma Function

- What is a “Sigma” function: Our sigma function is one that takes a positive integer value **n** and adds all the integers from **0 to n**.
  - n=0 then Sigma(0) = 0
  - n=1 then Sigma(1) = 1
  - n=2 then Sigma(2) = 2 + 1
  - n=4 then Sigma(4) = 4 + 3 + 2 + 1

```
int result = 0;

for (int i=1; i<=n; i++){

    result = result + i;
}
```

- Now lets make this into a method

The mySigma method

```
public static int mySigma(int n) {

   int result = 0;

   for (int i=1; i<=n; i++) {

      result = result + i;
   }

    return result;
}
```

Using the mySigma method

- Now, the rest of my Java program can use this mySigma() method.

– Such as the main() method or in any other methods we write

```
public static void main(){
...

 int x = mySigma(4);


..
}
```

What happens when we call the method mySigma?

1 A memory resource called a **stack frame or run time stack** is created for the call to the method int x = **mySigma(4**)

2 Memory space is set aside within this **stack frame** for the formal parameter **int n**

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

4 The value in the actual parameter (**4**) is <u>**copied**</u> into the formal parameters **n**.

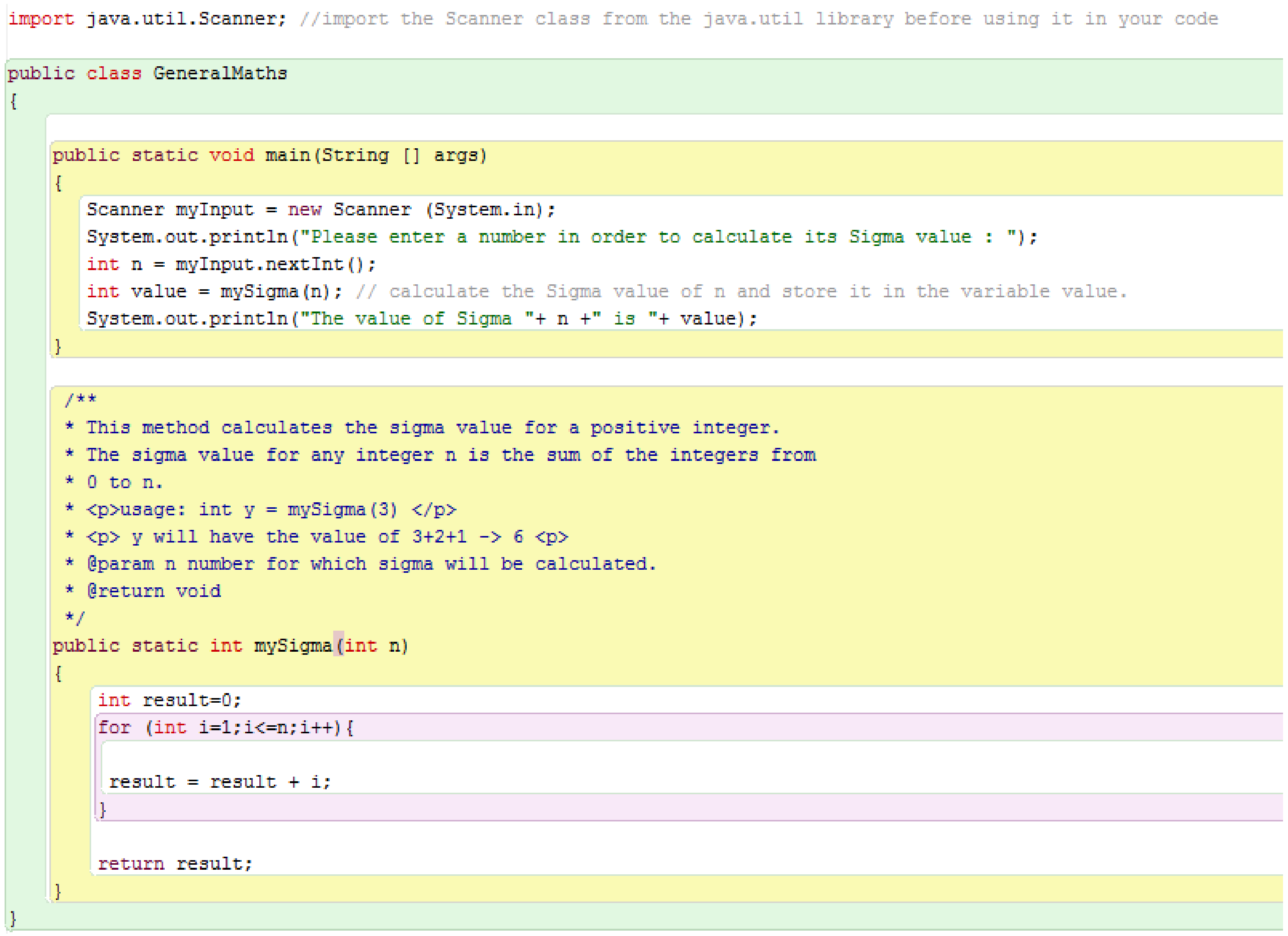
1 Space is set aside for the local variable **int result** and it is assigned the value **0.**

1 The code statements, which in this case is the for loop in the body of the method is now executed. The first thing that happens in this case is a local variable is declared **int i** whose scope is only for the duration of the for loop. (See scope rules earlier)

2 When the loop is finished executing, the variable **result** will have the calculated sigma result and this value will be **returned** to the point at which the method mySigma was called (**variable x** is given the returned value). The flow of control of the program returns to this point using the **return address** stored in the **stack frame**.

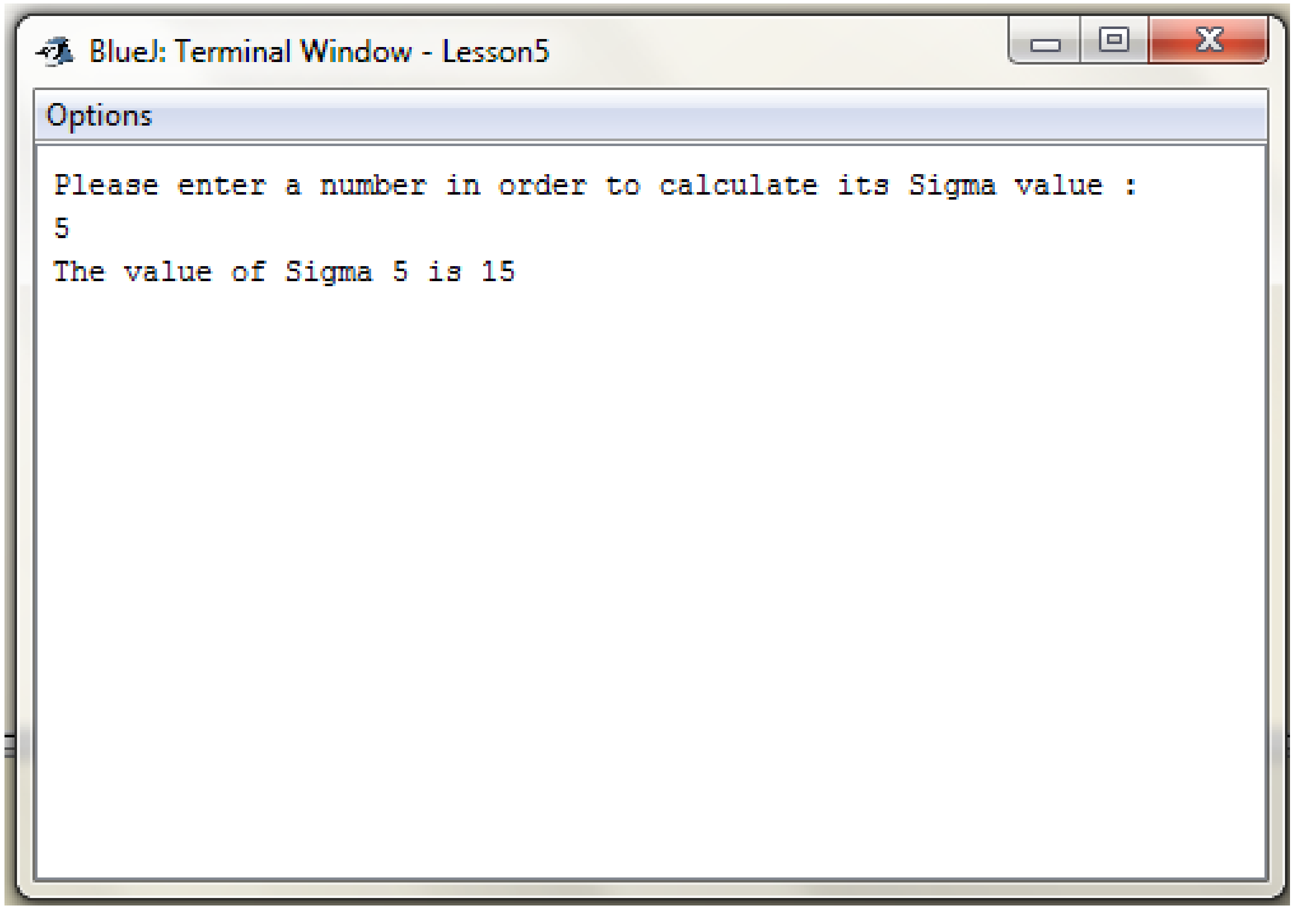
8 Finally the stack frame is **"destroyed"** or returned to the system to be reused. The Formal parameters **int n** as well as the local variable **int i** are no longer available in your program

Calling mySigma from main within the GeneralMaths Class



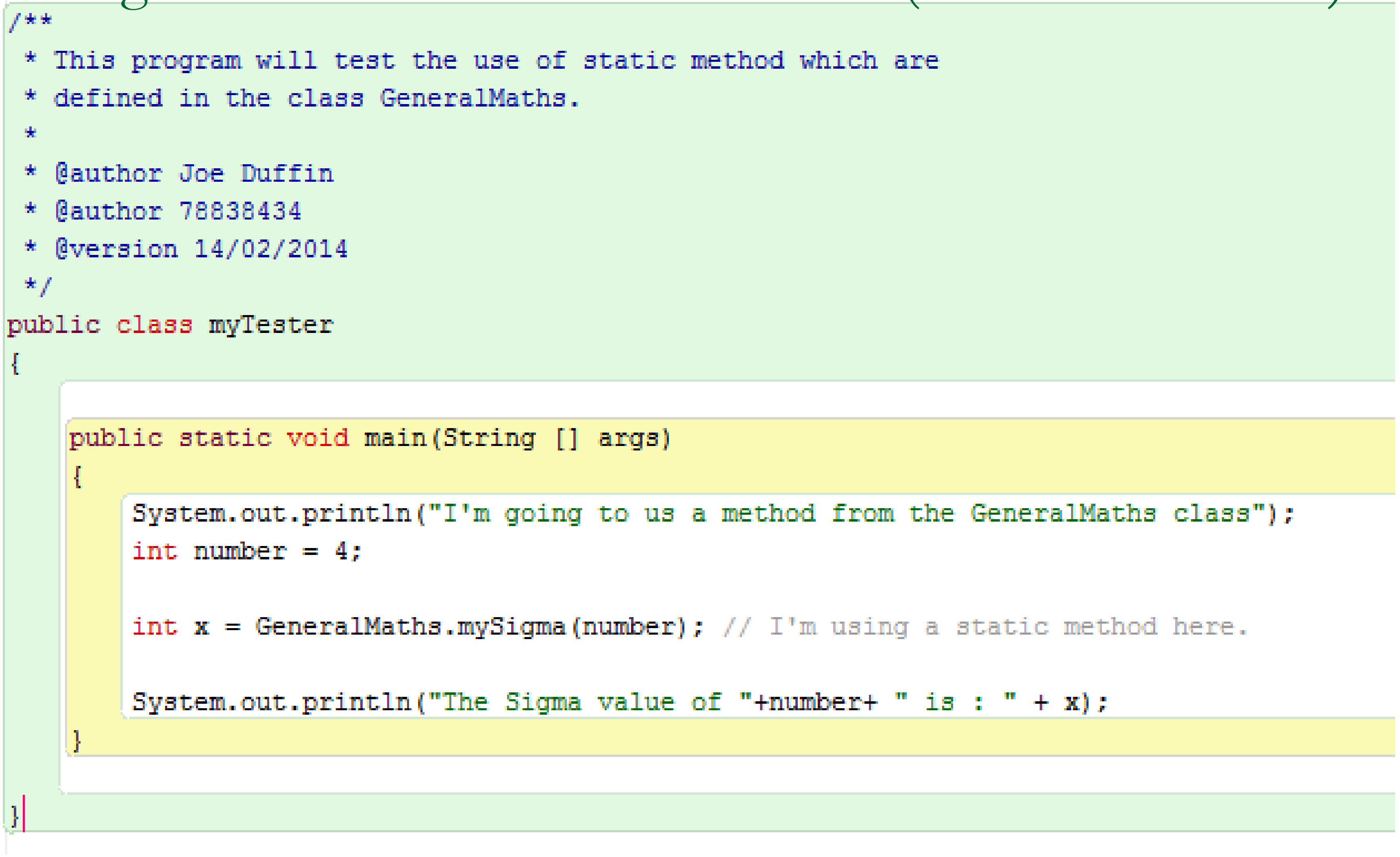
```
import java.util.Scanner; //import the Scanner class from the java.util library before using it in your code

public class GeneralMaths
{

    public static void main(String [] args)
    {
        Scanner myInput = new Scanner (System.in);
        System.out.println("Please enter a number in order to calculate its Sigma value : ");
        int n = myInput.nextInt();
        int value = mySigma(n); // calculate the Sigma value of n and store it in the variable value.
        System.out.println("The value of Sigma "+ n +" is "+ value);
    }

     /**
     * This method calculates the sigma value for a positive integer.
     * The sigma value for any integer n is the sum of the integers from
     * 0 to n.
     * <p>usage: int y = mySigma(3) </p>
     * <p> y will have the value of 3+2+1 -> 6 <p>
     * @param n number for which sigma will be calculated.
     * @return void
     */
    public static int mySigma(int n)
    {
        int result=0;
        for (int i=1;i<=n;i++){

         result = result + i;
        }

        return result;
    }
}
```

Output of main method call to mySigma in GeneralMaths



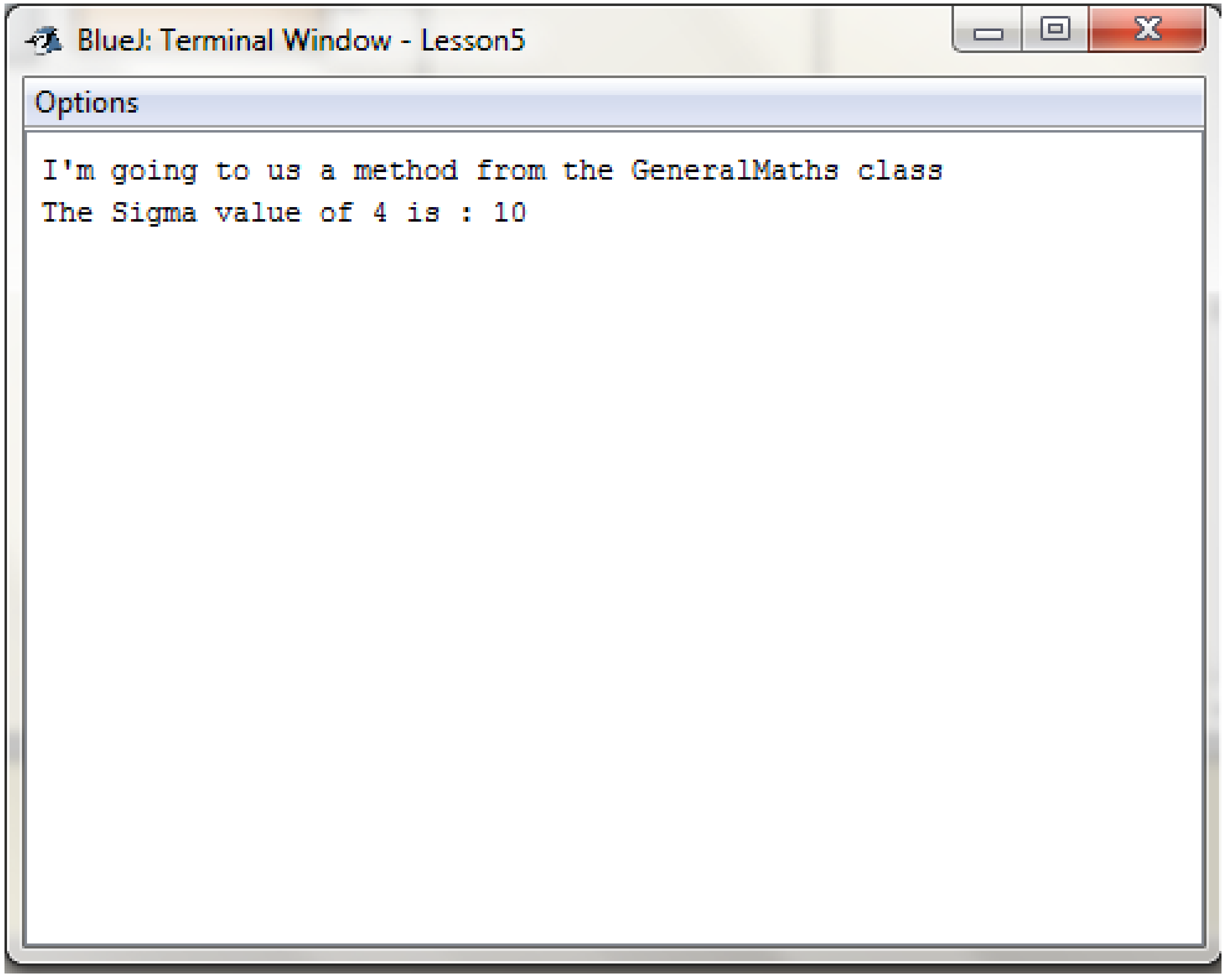
```
Please enter a number in order to calculate its Sigma value :
5
The value of Sigma 5 is 15
```

Calling mySigma method from another program (myTester) Calling the method from another class (static method call)



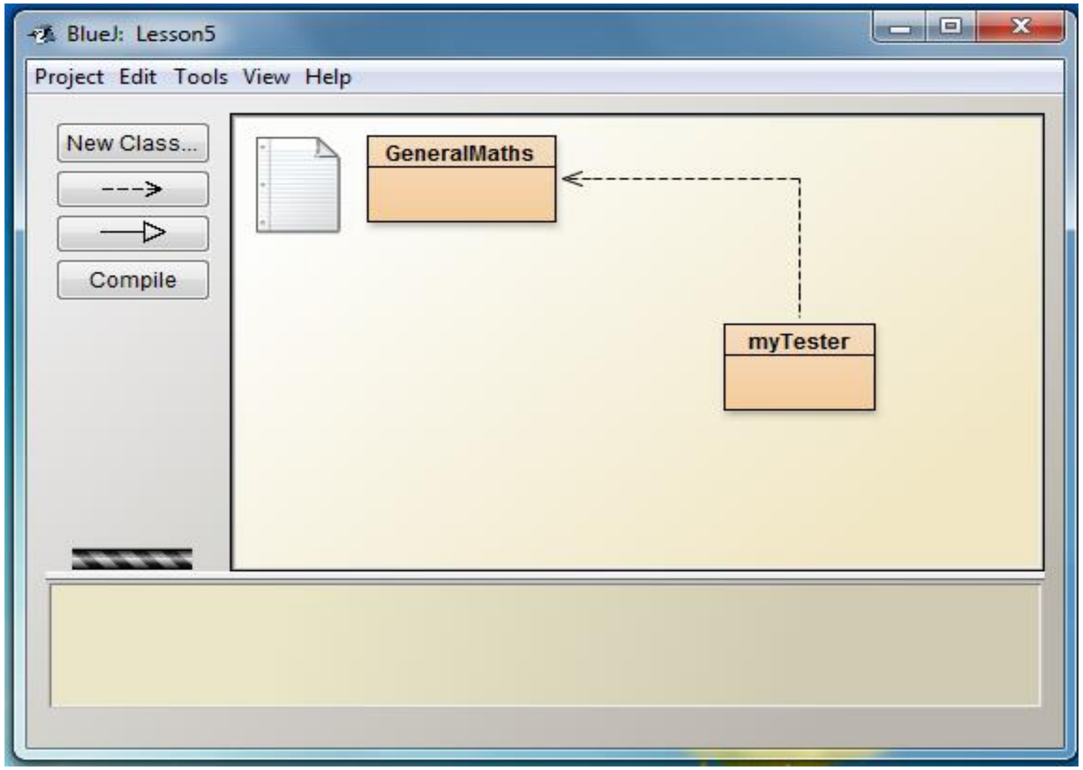
```
/**
 * This program will test the use of static method which are
 * defined in the class GeneralMaths.
 *
 * @author Joe Duffin
 * @author 78838434
 * @version 14/02/2014
 */
public class myTester
{

    public static void main(String [] args)
    {
        System.out.println("I'm going to us a method from the GeneralMaths class");
        int number = 4;

        int x = GeneralMaths.mySigma(number); // I'm using a static method here.

        System.out.println("The Sigma value of "+number+ " is : " + x);
    }

}
```

Calling mySigma from another class (MyTester) using a static method call of the form: **GeneralMaths.MySigma(4)**



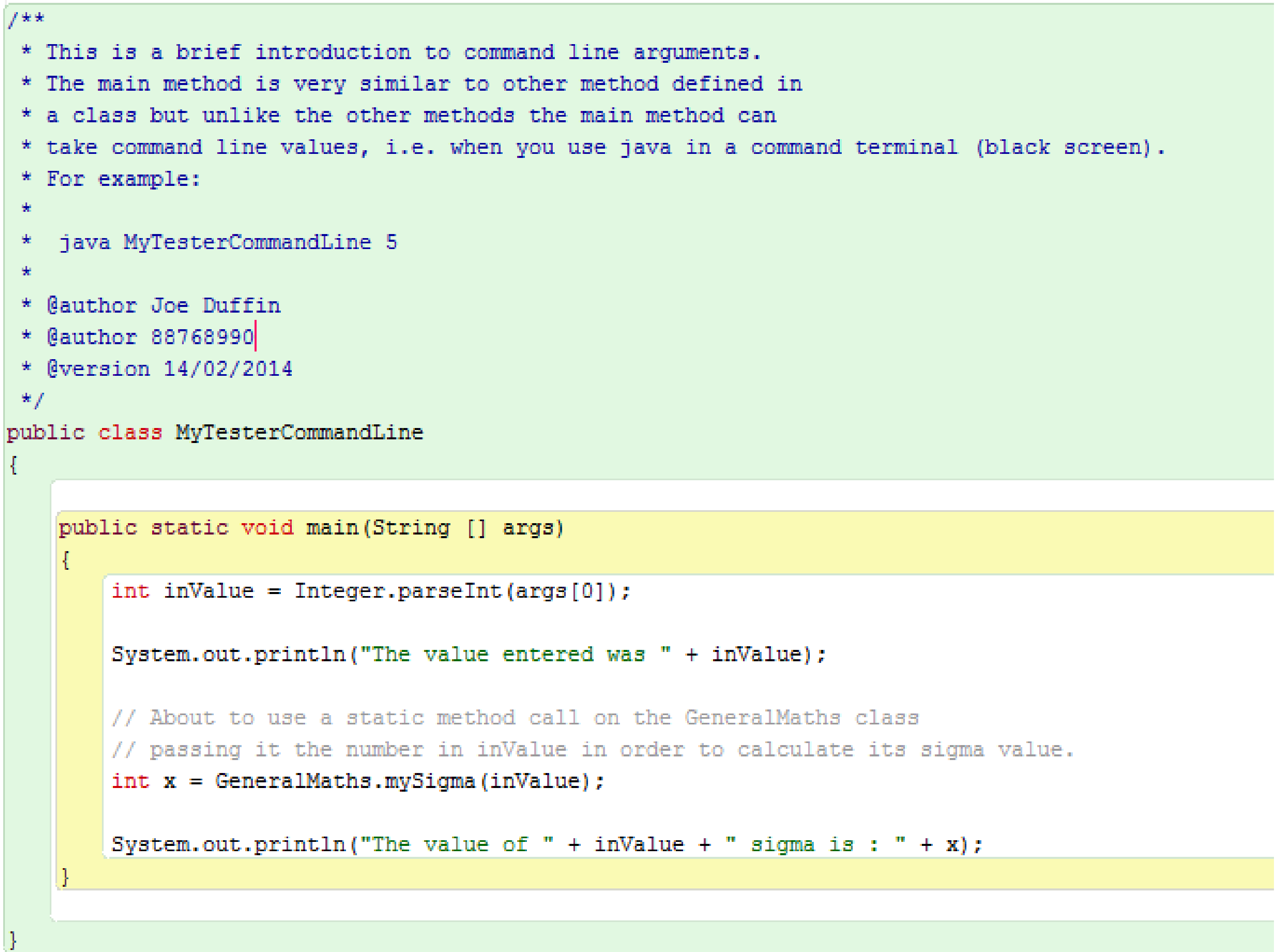
```
I'm going to us a method from the GeneralMaths class
The Sigma value of 4 is : 10
```

UML (Unified Modelling Language), box and arrow diagrams used in industry.



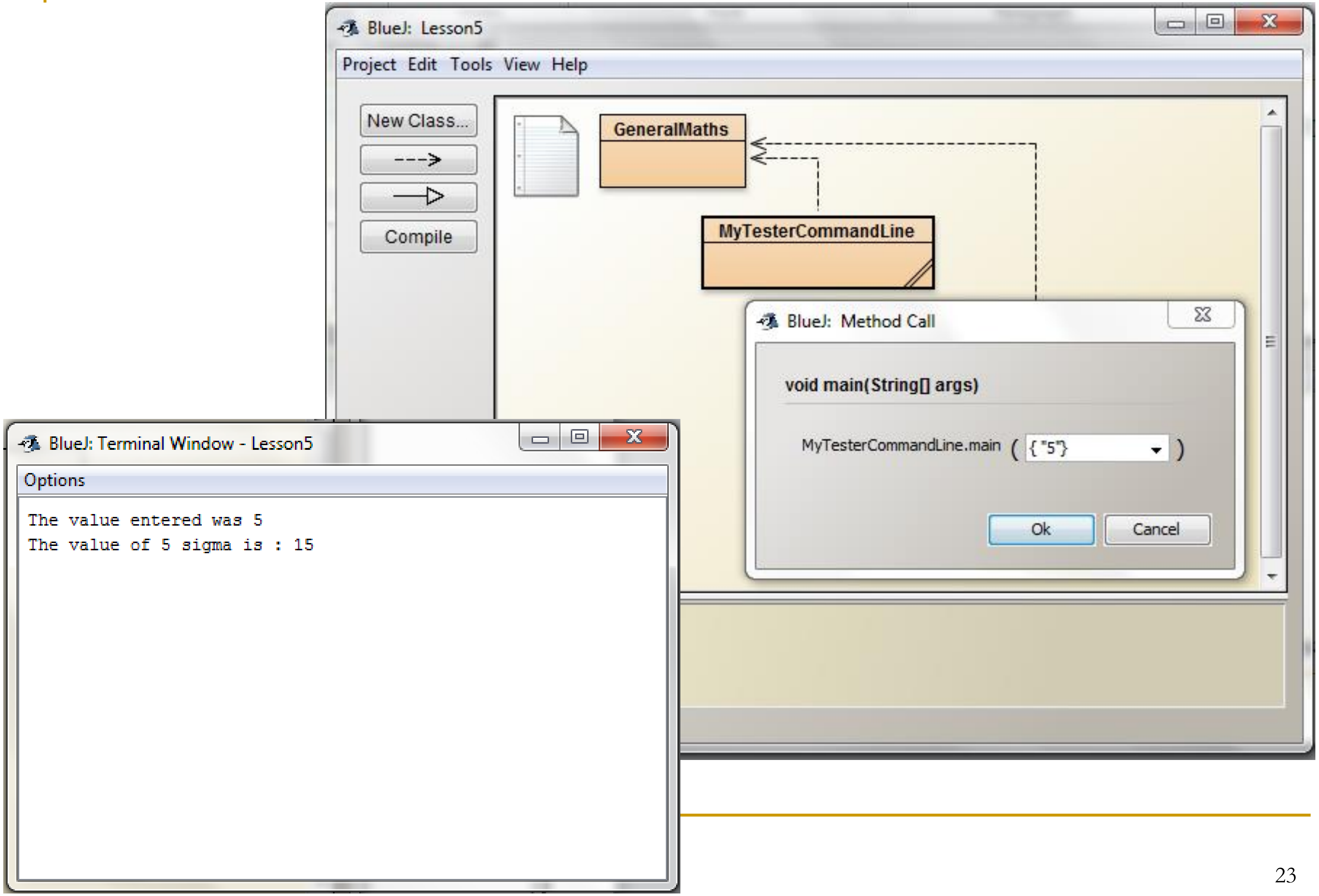
BlueJ shows the UML relationship between the myTester class and the GeneralMaths Class. The broken arrow indicates that myTester uses some aspect of the GeneralMaths class. (i.e. the mySigma method)

Passing a value to main in BlueJ and using the command line. The pass this value to **mySigma** method in **GeneralMaths**



```
/**
 * This is a brief introduction to command line arguments.
 * The main method is very similar to other method defined in
 * a class but unlike the other methods the main method can
 * take command line values, i.e. when you use java in a command terminal (black screen).
 * For example:
 *
 *  java MyTesterCommandLine 5
 *
 * @author Joe Duffin
 * @author 88768990
 * @version 14/02/2014
 */
public class MyTesterCommandLine
{

    public static void main(String [] args)
    {
        int inValue = Integer.parseInt(args[0]);

        System.out.println("The value entered was " + inValue);

        // About to use a static method call on the GeneralMaths class
        // passing it the number in inValue in order to calculate its sigma value.
        int x = GeneralMaths.mySigma(inValue);

        System.out.println("The value of " + inValue + " sigma is : " + x);
    }

}
```

Running the command line program BlueJ

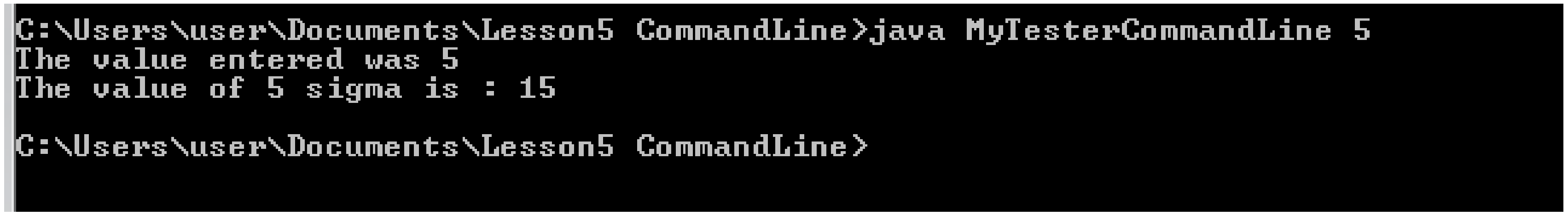


void main(String[] args)

MyTesterCommandLine.main ({"5"})

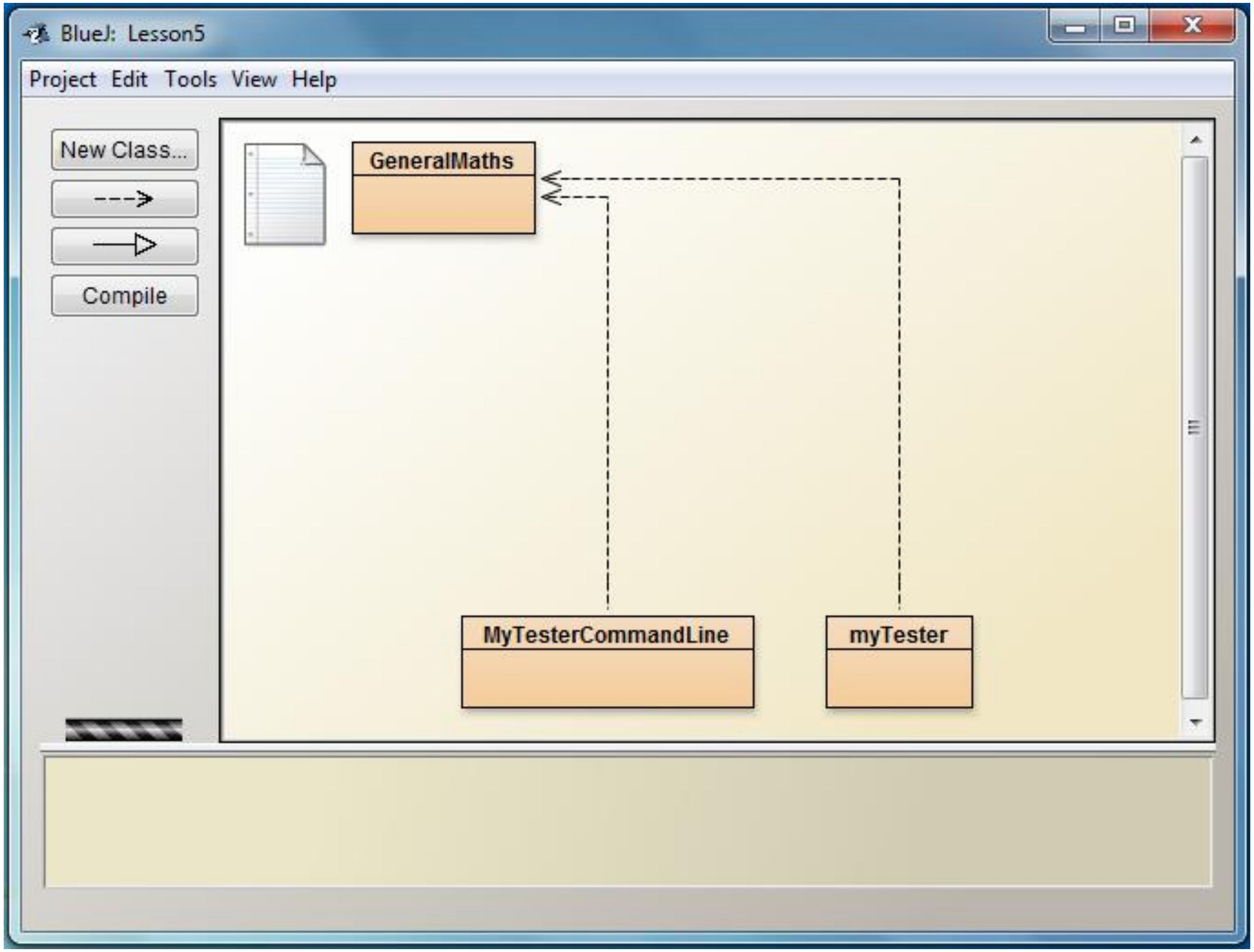
```
The value entered was 5
The value of 5 sigma is : 15
```

Running MyTesterCommandLine using the java command



```
C:\Users\user\Documents\Lesson5 CommandLine>java MyTesterCommandLine 5
The value entered was 5
The value of 5 sigma is : 15

C:\Users\user\Documents\Lesson5 CommandLine>
```

UML showing two classes using the GeneralMaths class


BlueJ: Lesson5
Project Edit Tools View Help
New Class...
Compile
GeneralMaths
MyTesterCommandLine
myTester