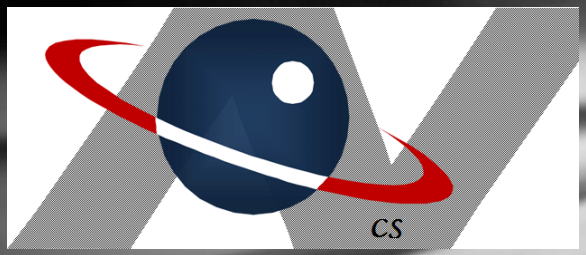
O(N) CS LESSONS

Lesson 5C – MyClass Methods



By John B. Owen

Table of Contents

Objectives

Students will:

- Develop their own class of utility methods called MyClass, writing stand-alone "static" methods, similar to those found in the Math class, but dealing with a variety of tasks.
- Learn to create their own API using the JavaDoc utility, making a help file for their customized class.

CodingBat.com

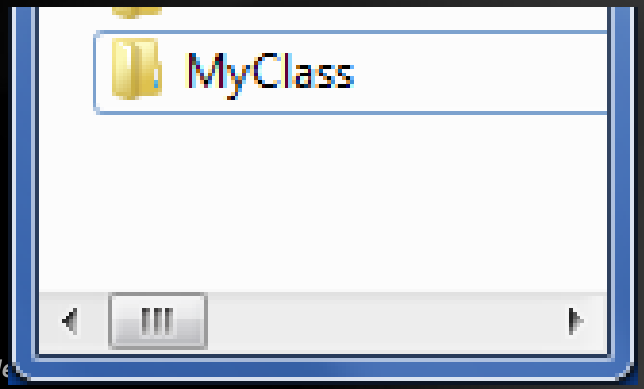
- The AND, OR and NOT boolean operators are briefly introduced at the end of the lesson in some special labs using a website called codingbat.com.

Defining your own class

- Every program in Java is a class.
- For most of what you have done so far, the class has only contained one method, the **main method**, in which all of your work has been done, and which is **required** to run most programs.
- Now you will add more methods to your class that can do various tasks.

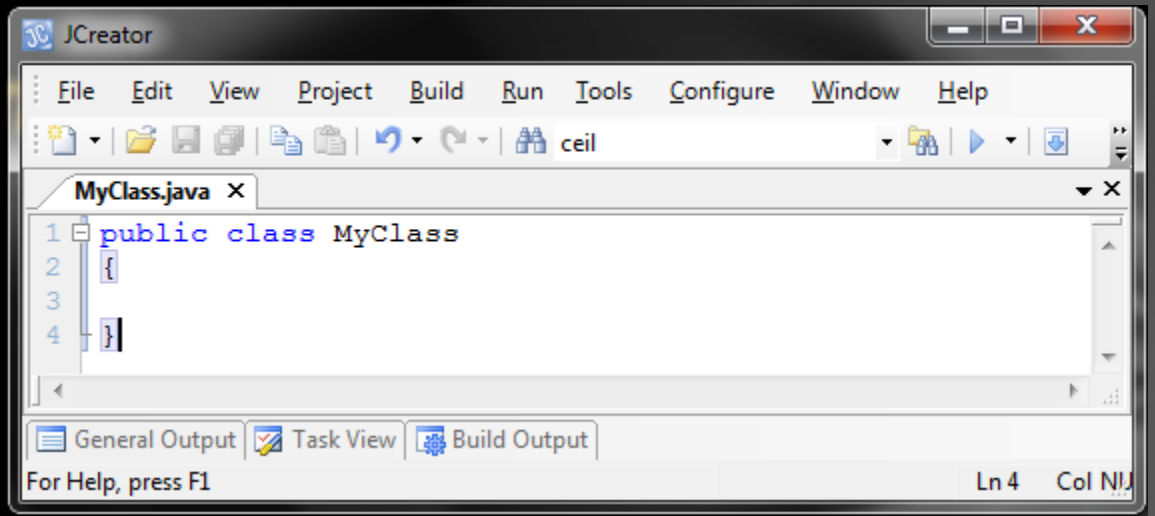
Create a separate folder

- It is good practice to create a separate folder for each lesson you do, although not essential.
- Make one called MyClass for all the work in this lesson, and save all of your Java files to this folder.



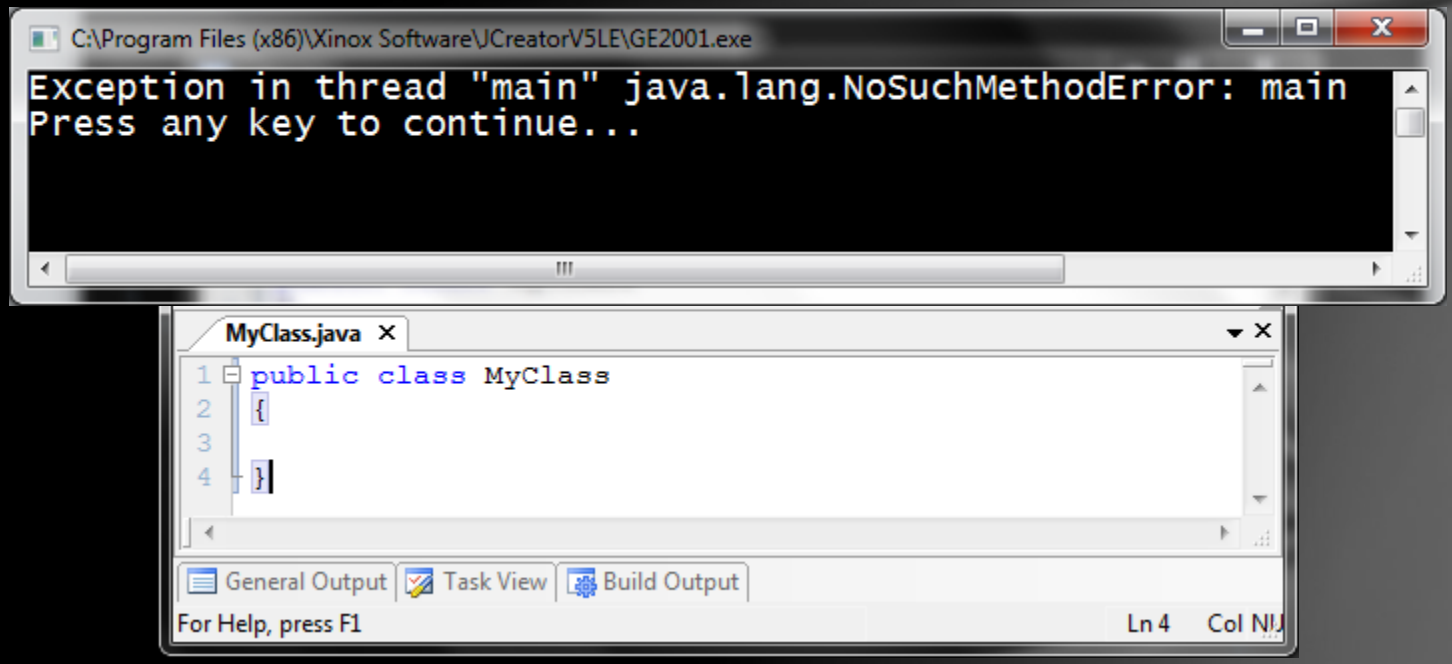
The class skeleton

- We'll start with the basic skeleton of the class, which essentially contains nothing, as you can see.
- *Be sure to save this class file in the MyClass folder you just created.*



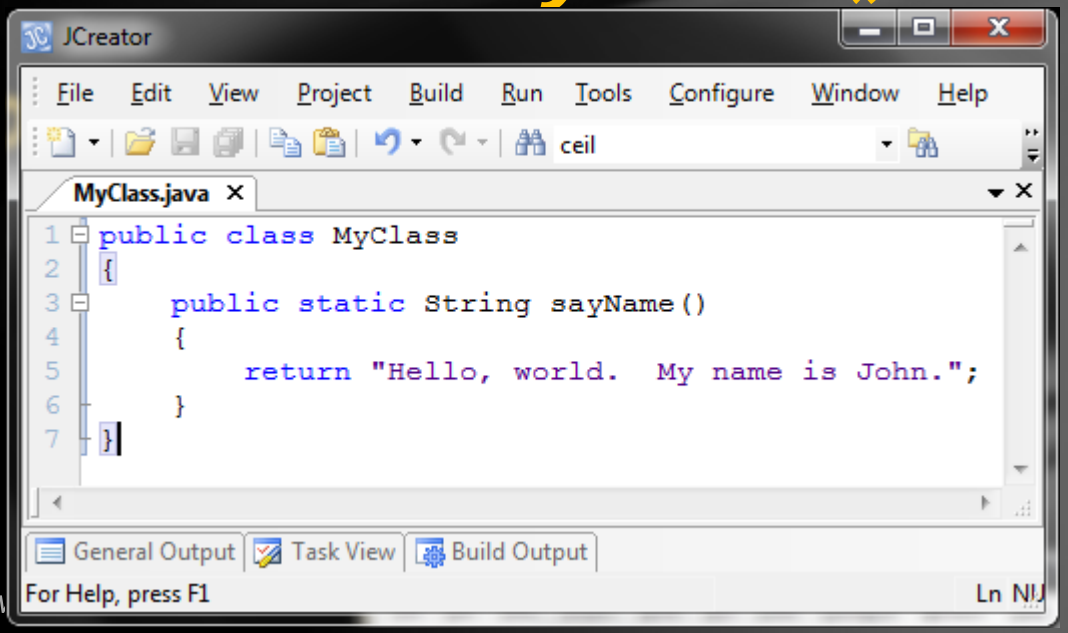
The class skeleton

- It will compile, but will not run since there is no main method, but that is OK for now..



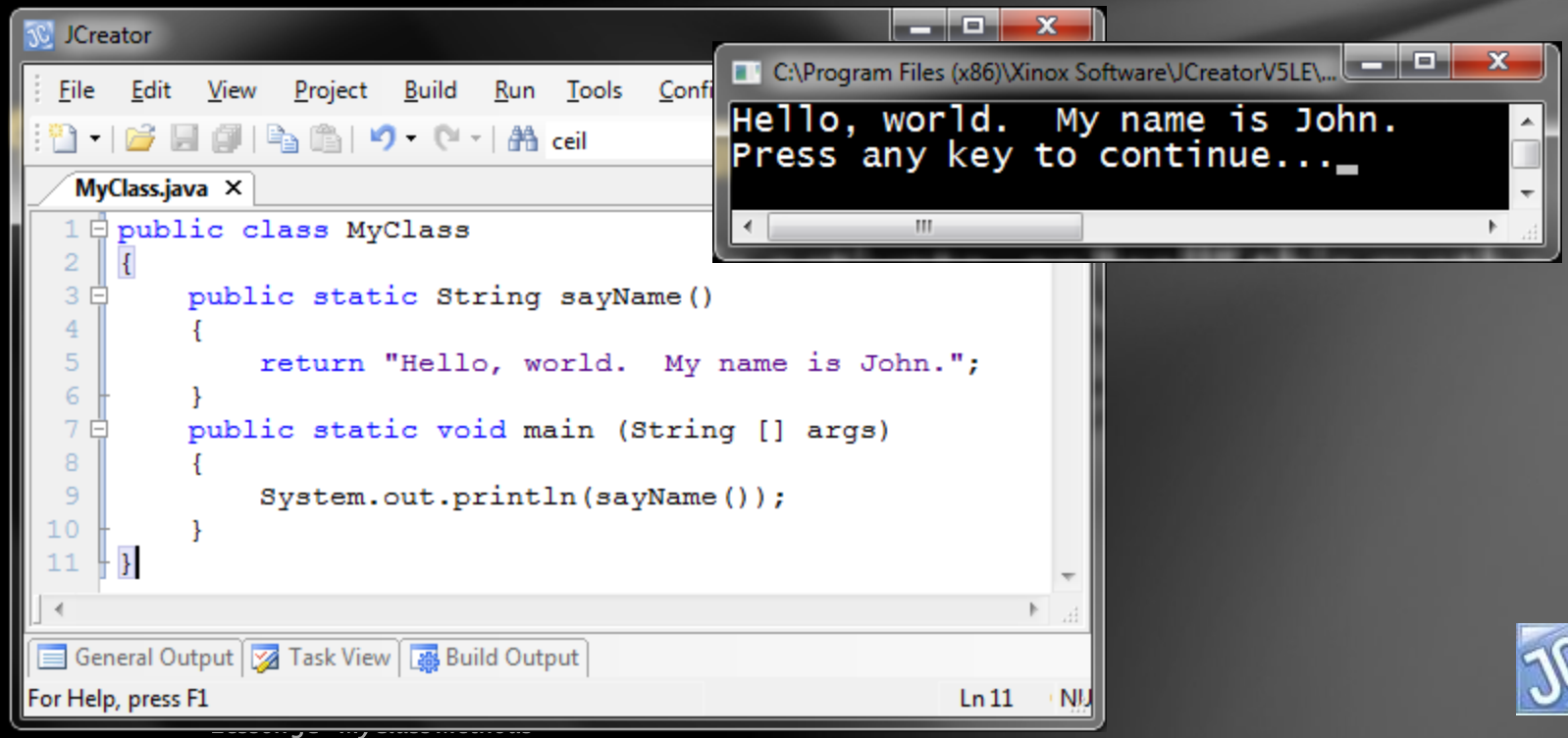
Defining and calling a static method

- Let's define a very simple static method, one that returns a String, like, "Hello, world. My name is John."
- We'll call the method ***sayName()***
- ***Here it is!***



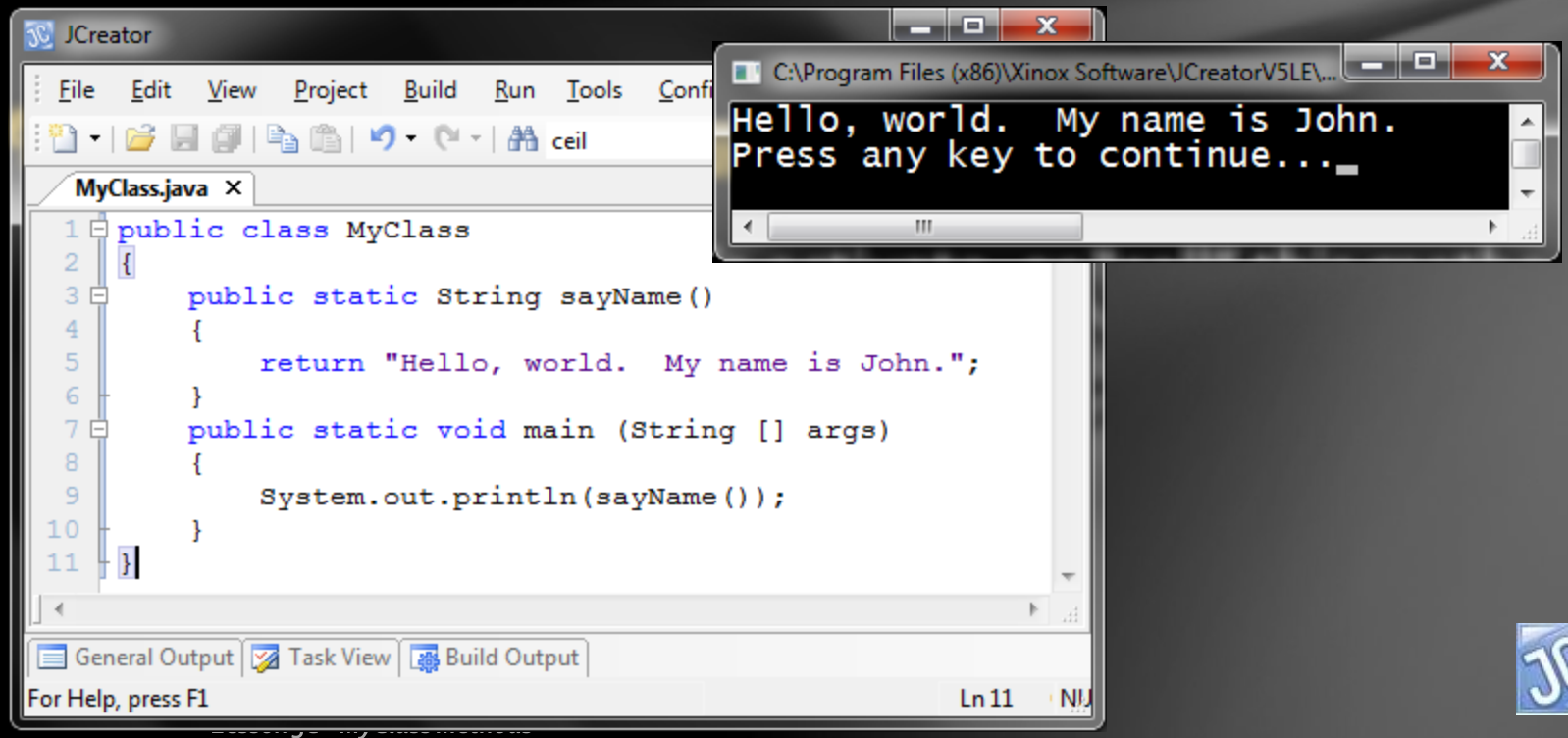
Defining and calling a static method

- To activate, or "call" this method, we need the main method with an output statement.



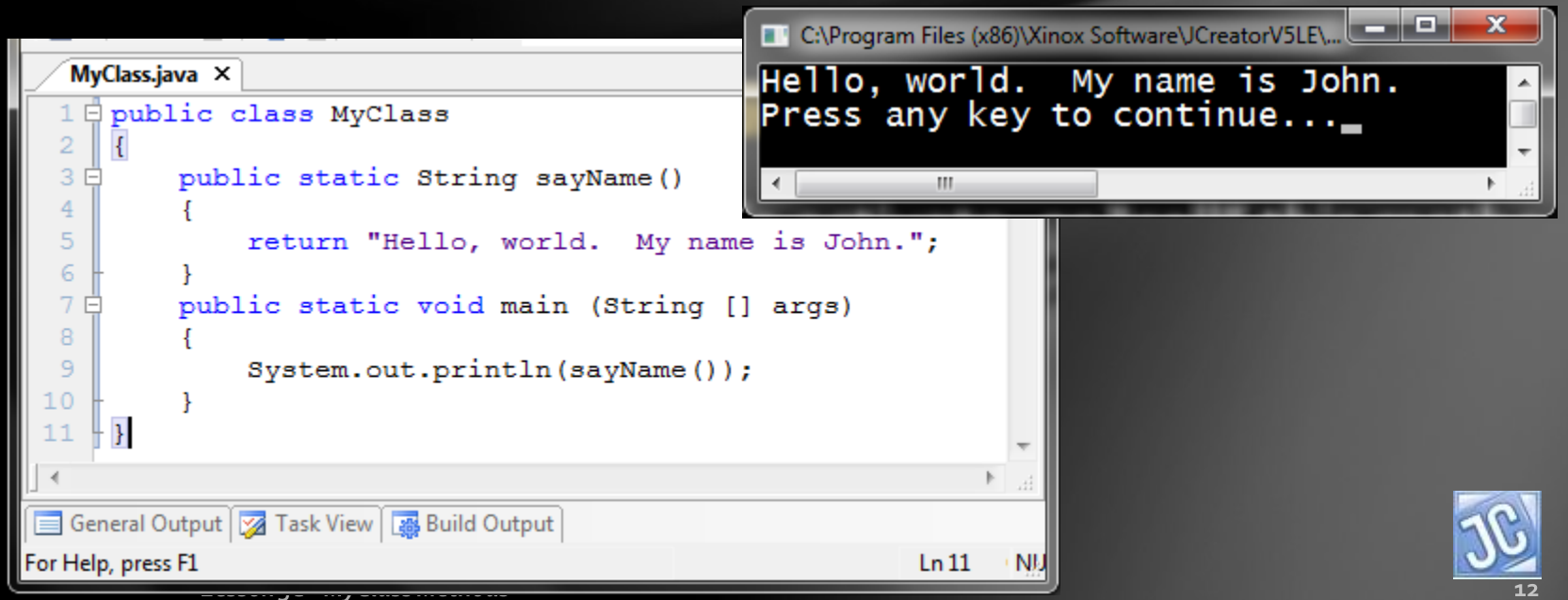
Defining and calling a static method

- Since the println method requires a string, the call to ***sayName()*** provides that String.



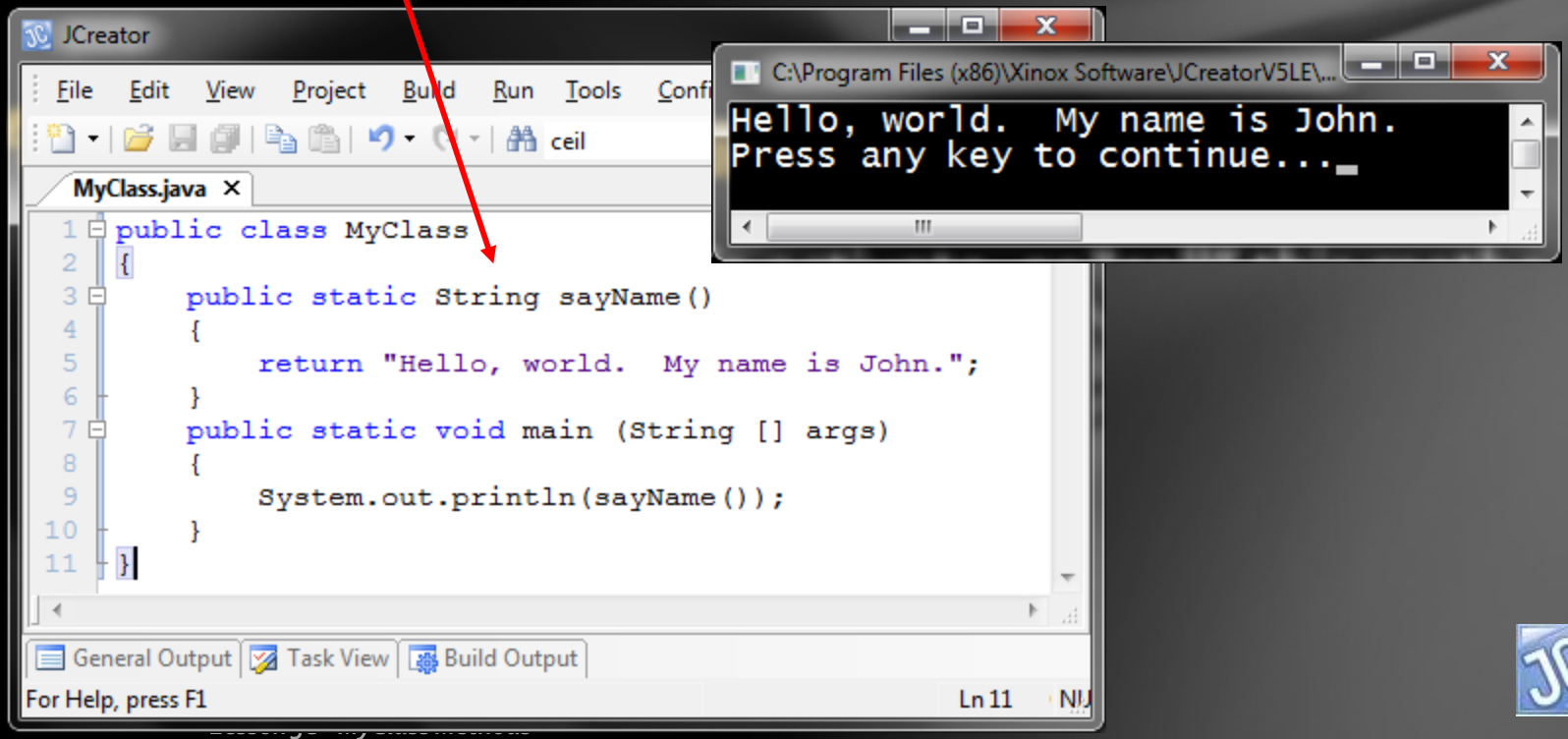
Defining and calling a static method

- When the ***sayName()*** method is called, the body of the method is executed, and a String is returned to the source of the call.



Method structure - sayName

- Let's talk about the method structure.
- This is a *return* method since it returns something, in this case, a String.



Method structure - sayName

- The method header contains four parts: public static String sayName()

```
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
```

Method structure - sayName

- The "public" designation makes the method available to use anywhere the class is "visible" (more on that later).

```
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
```

- There are two other access designations we'll also discuss later on: ***private*** and ***protected***

Method structure - sayName

- The word "static" makes it a "stand-alone" method, belonging to the class, not requiring an object to call it.
- This is like a Math class method because it is "stand-alone", but not like a String class method, which requires an object.

```
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
```

Method structure - sayName

- The word "String" indicates the type of data it will return.
- Finally, the name of the method, ***sayName()***, completes the header as the *identifier* of the method.

```
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
```

Method structure - sayName

- The empty parentheses are necessary to indicate to the compiler that it is a *method*, even though no parameter (variable inside parentheses) is required for this one.

```
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
```

Method structure - sayName

- The body of the method is contained inside the pair of curly braces… { }
- For this method, the only statement in the body is the return statement, which returns the String.

```
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
```

Method structure - sayName

- One final but important part is the documentation, which is a simple comment that precedes the method definition, but is important, especially if a method is particularly complex.

```
//returns a greeting sentence
public static String sayName()
{
    return "Hello, world.  My name is John.";
}
public static void main (String [] args)
{
    System.out.println(sayName());
}
```

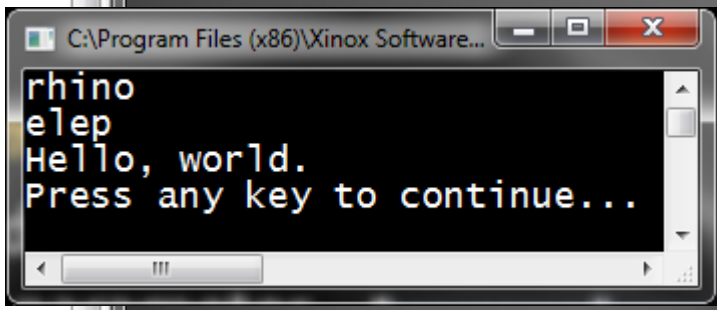
Parameter method - firstHalf

- Now let's define a method that requires a parameter.

MyClass.java

```
1  public class MyClass
2  {
3      //returns a greeting sentence
4      public static String sayName()
5      {
6          return "Hello, world.  My name is John.";
7      }
8      //returns the first half of the parameter word
9      public static String firstHalf(String word)
10     {
11         int halfway = word.length()/2;
12         String answer = word.substring(0,halfway);
13         return answer;
14     }
15     public static void main (String [] args)
16     {
17         //System.out.println(sayName());
18         System.out.println(firstHalf("rhinoceros"));
19         System.out.println(firstHalf("elephants"));
20         System.out.println(firstHalf(sayName()));
21     }
22 }
```

Study carefully this new method definition for ***firstHalf***, and the output shown below.

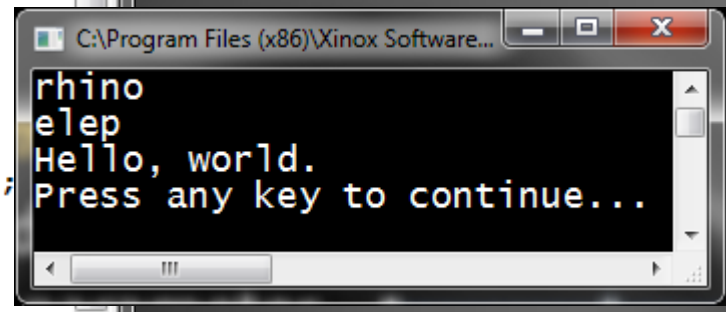


Parameter method - firstHalf

MyClass.java

```
1  public class MyClass
2  {
3      //returns a greeting sentence
4      public static String sayName()
5      {
6          return "Hello, world.  My name is John.";
7      }
8      //returns the first half of the parameter word
9      public static String firstHalf(String word)
10     {
11         int halfway = word.length()/2;
12         String answer = word.substring(0,halfway);
13         return answer;
14     }
15     public static void main (String [] args)
16     {
17         //System.out.println(sayName());
18         System.out.println(firstHalf("rhinoceros"));
19         System.out.println(firstHalf("elephants"));
20         System.out.println(firstHalf(sayName()));
21     }
22 }
```

The ***firstHalf*** method will receive a String parameter, calculate the first half of it, and then returns that calculated answer.



Parameter method - firstHalf

MyClass.java

```
1  public class MyClass
2  {
3      //returns a greeting sentence
4      public static String sayName()
5      {
6          return "Hello, world.  My name is John.";
7      }
8      //returns the first half of the parameter word
9      public static String firstHalf(String word)
10     {
11         int halfway = word.length()/2;
12         String answer = word.substring(0,halfway);
13         return answer;
14     }
15     public static void main (String [] args)
16     {
17         //System.out.println(sayName());
18         System.out.println(firstHalf("rhinoceros"));
19         System.out.println(firstHalf("elephants"));
20         System.out.println(firstHalf(sayName()));
21     }
22 }
```

Notice carefully that other String class methods were used in the definition of the new custom method.

Using previously defined methods is an important technique and should be done as often as possible to save work and avoid duplication of effort.

```
C:\Program Files (x86)\Xinox Software...
rhino
elep
Hello, world.
Press any key to continue...
```

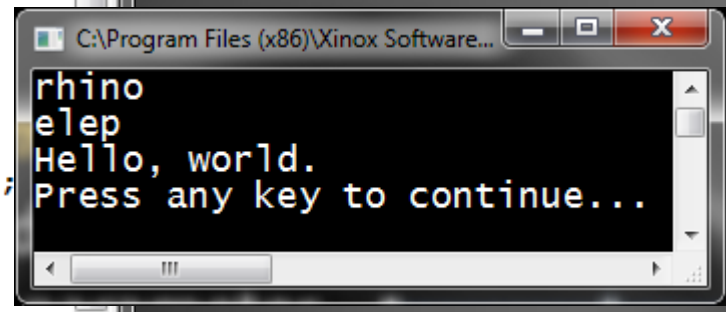
Parameter method - firstHalf

MyClass.java

```
1  public class MyClass
2  {
3      //returns a greeting sentence
4      public static String sayName()
5      {
6          return "Hello, world.  My name is John.";
7      }
8      //returns the first half of the parameter word
9      public static String firstHalf(String word)
10     {
11         int halfway = word.length()/2;
12         String answer = word.substring(0,halfway);
13         return answer;
14     }
15     public static void main (String [] args)
16     {
17         //System.out.println(sayName());
18         System.out.println(firstHalf("rhinoceros"));
19         System.out.println(firstHalf("elephants"));
20         System.out.println(firstHalf(sayName()));
21     }
22 }
```

Also notice that each calculation step was done separately.

Although it is possible to imbed all the steps into one statement, it makes your code clearer if you limit the actions to one or two steps per command.



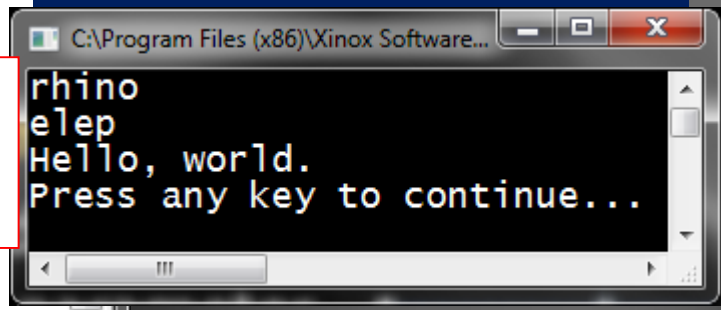
Parameter method - firstH

MyClass.java

```
1  public class MyClass
2  {
3      //returns a greeting sentence
4      public static String sayName()
5      {
6          return "Hello, world.  My name is John.";
7      }
8      //returns the first half of the parameter word
9      public static String firstHalf(String word)
10     {
11         int halfway = word.length()/2;
12         String answer = word.substring(0,halfway);
13         return answer;
14     }
15
16     //returns the first half of the parameter word
17     public static String firstHalf(String word)
18     {
19         return word.substring(0,word.length()/2);
20     }
21     System.out.println(firstHalf(sayName()));
22 }
```

Just in case you are curious, below is the "condensed" version of the method, where everything ***is*** done in one command.

The same steps are all there, just in one longer command, which in this case is not too bad, but is just about as long as you would want and still have good programming style.



Math method - firstLastValue

- Now let's define a method that calculates and returns an integer.
- We'll call it ***firstLastValue(String str)***, which will receive a String, add together the ASCII values of the first and last letters in the String, and return that total.

Math method - firstLastValue

Here it is!

Again, the steps are all calculated separately, with the final answer returned as a simple addition step.

JCreator

File Edit View Project Build Run Tools Configure Window Help

MyClass.java

```
10      {
11          return word.substring(0,word.length()/2);
12      }
13      //returns the ASCII value total of the first
14      //and last characters of str
15      public static int firstLastValue(String str)
16      {
17          int first = str.charAt(0);
18          int last = str.charAt(str.length()-1);
19          return first+last;
20      }
21      public static void main (String [] args)
22      {
23          //System.out.println(sayName());
24          System.out.println(firstLastValue("rhinoceros"));
25          System.out.println(firstLastValue("elephants"));
26          System.out.println(firstLastValue(sayName()));
27      }
28  }
```

General Output | Task View | Build Output

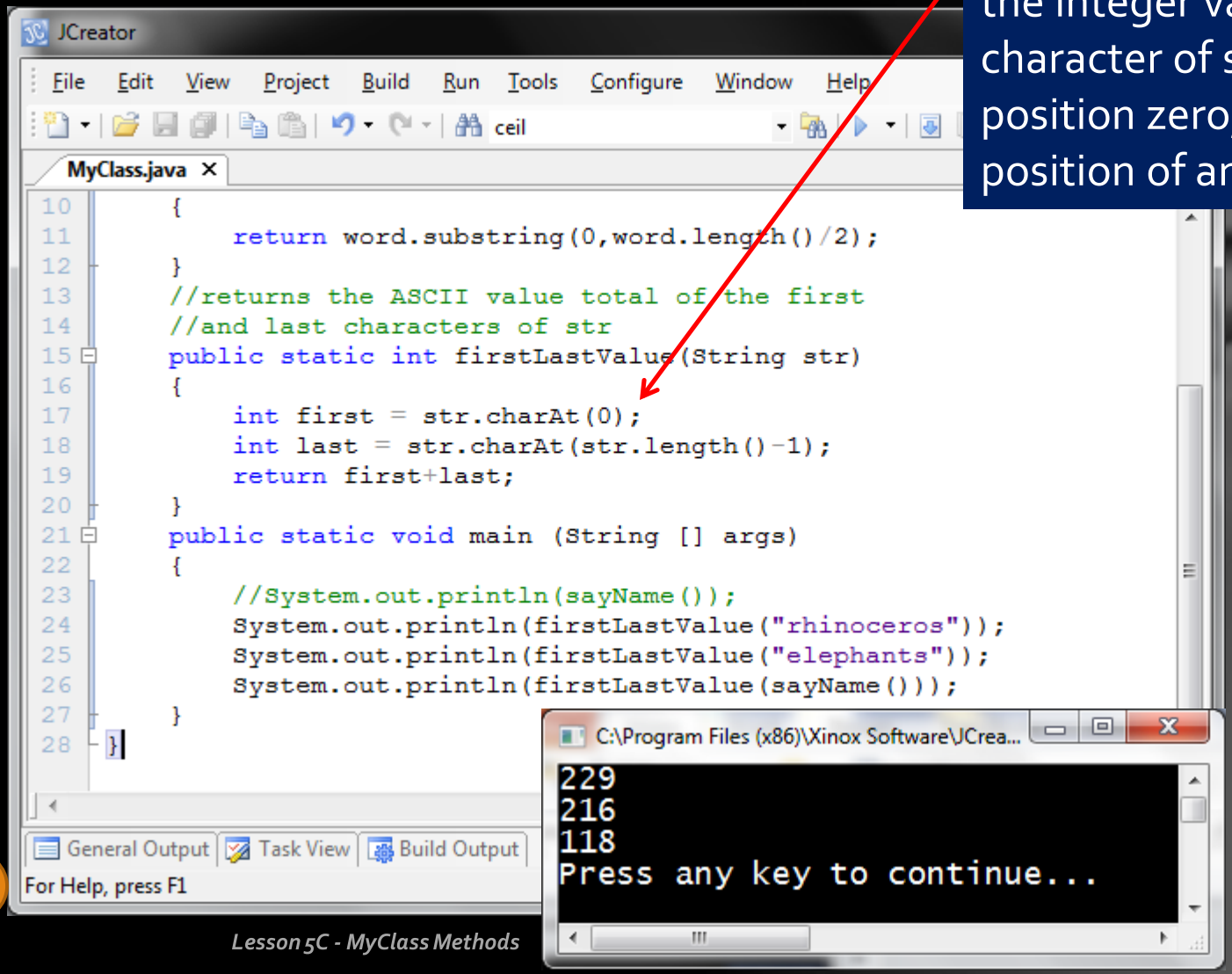
For Help, press F1

C:\Program Files (x86)\Xinox Software\JCrea...

```
229
216
118
Press any key to continue...
```

Math method - firstLastValue

The first calculation finds the integer value of the first character of str, the one at position zero, the first position of any string.



Math method - firstLastValue

Since characters ARE ***integers in disguise***, they are "compatible" with and can be assigned to integer variables.

When the character is assigned to the integer variable, its ASCII value is actually what is stored.

JCreator

File Edit View Project Build Run Tools Configure Window Help

ceil

MyClass.java

```
10      {
11          return word.substring(0,word.length()/2);
12      }
13      //returns the ASCII value total of the first
14      //and last characters of str
15      public static int firstLastValue(String str)
16      {
17          int first = str.charAt(0);
18          int last = str.charAt(str.length()-1);
19          return first+last;
20      }
21      public static void main (String [] args)
22      {
23          //System.out.println(sayName());
24          System.out.println(firstLastValue("rhinoceros"));
25          System.out.println(firstLastValue("elephants"));
26          System.out.println(firstLastValue(sayName()));
27      }
28  }
```

General Output | Task View | Build Output

For Help, press F1

C:\Program Files (x86)\Xinox Software\JCrea...

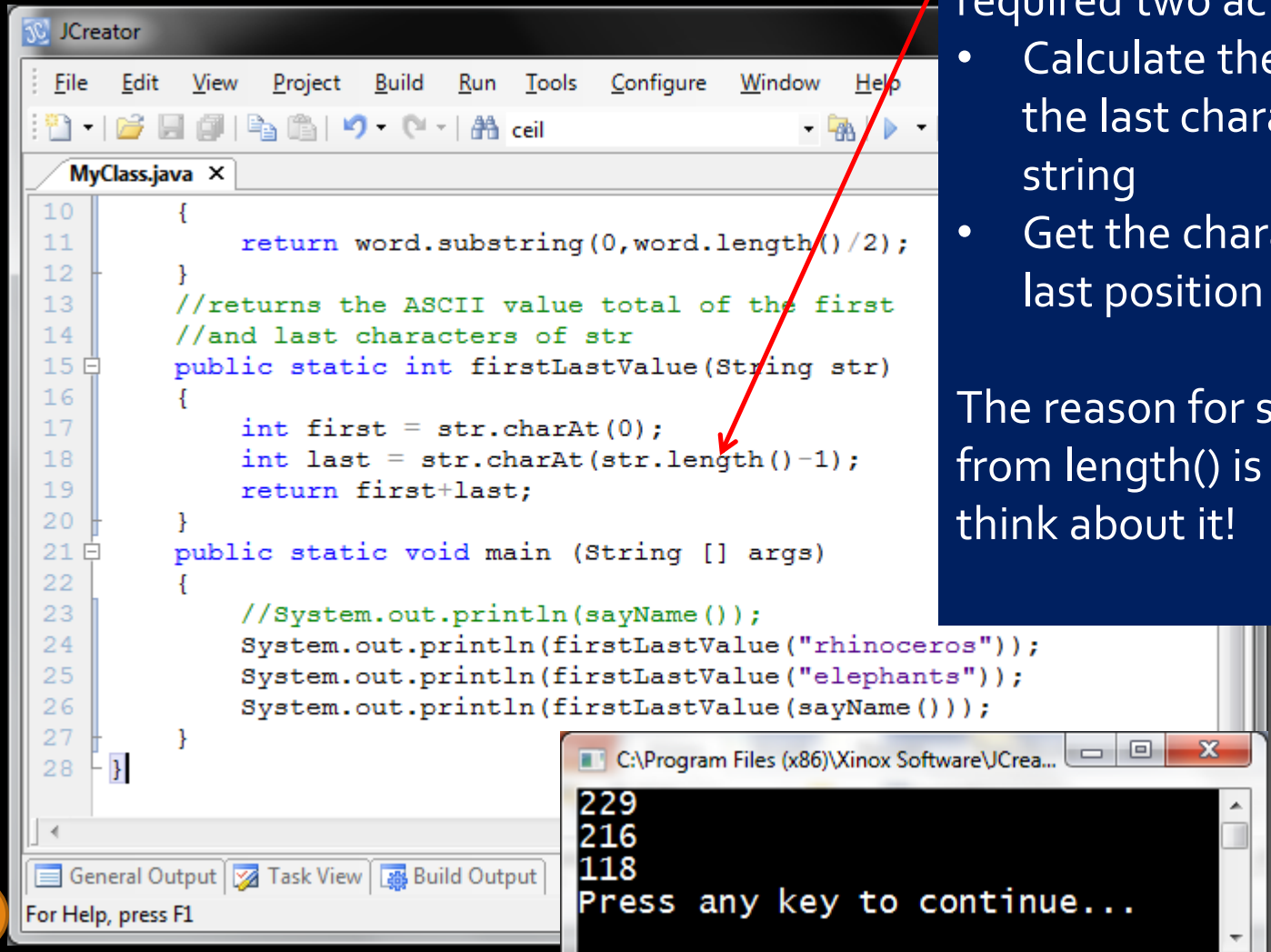
```
229
216
118
Press any key to continue...
```

Math method - firstLastValue

The second calculation required two actions:

- Calculate the position of the last character in the string
- Get the character at that last position

The reason for subtracting 1 from length() is important… think about it!



Math method - firstLastValue

Here's why:

- For the string "rhinoceros", the length command returns the value **10**, the number of characters in this string.
- However, the last letter of "rhinoceros" is in position 9 because of "zero-indexing" *(we start counting at zero).*
- Therefore, for any string, the last position is always its length, minus 1.

JCreator

File Edit View Project Build Run Tools Configure Window Help

MyClass.java

```
10      {
11          return word.substring(0,word.length()/2);
12      }
13      //returns the ASCII value total of the first
14      //and last characters of str
15      public static int firstLastValue(String str)
16      {
17          int first = str.charAt(0);
18          int last = str.charAt(str.length()-1);
19          return first+last;
20      }
21      public static void main (String [] args)
22      {
23          //System.out.println(sayName());
24          System.out.println(firstLastValue("rhinocer
25          System.out.println(firstLastValue("elephant
26          System.out.println(firstLastValue(sayName()
27      }
28  }
```

General Output | Task View | Build Output

For Help, press F1

C:\Program Files (x86)\Xinox Software\JCrea...

```
229
216
118
Press any key to continue...
```

JavaDocs for JCreator

- Now it is time to create your own API, or "help" file, using a built-in Java tool called JavaDoc.
- The steps to customize your JCreator IDE for this capability are covered in the next few slides.
- Follow these steps VERY CAREFULLY!
  - javadoc - creates the help file
  - show javadoc - opens the help file

JavaDocs for JCreator

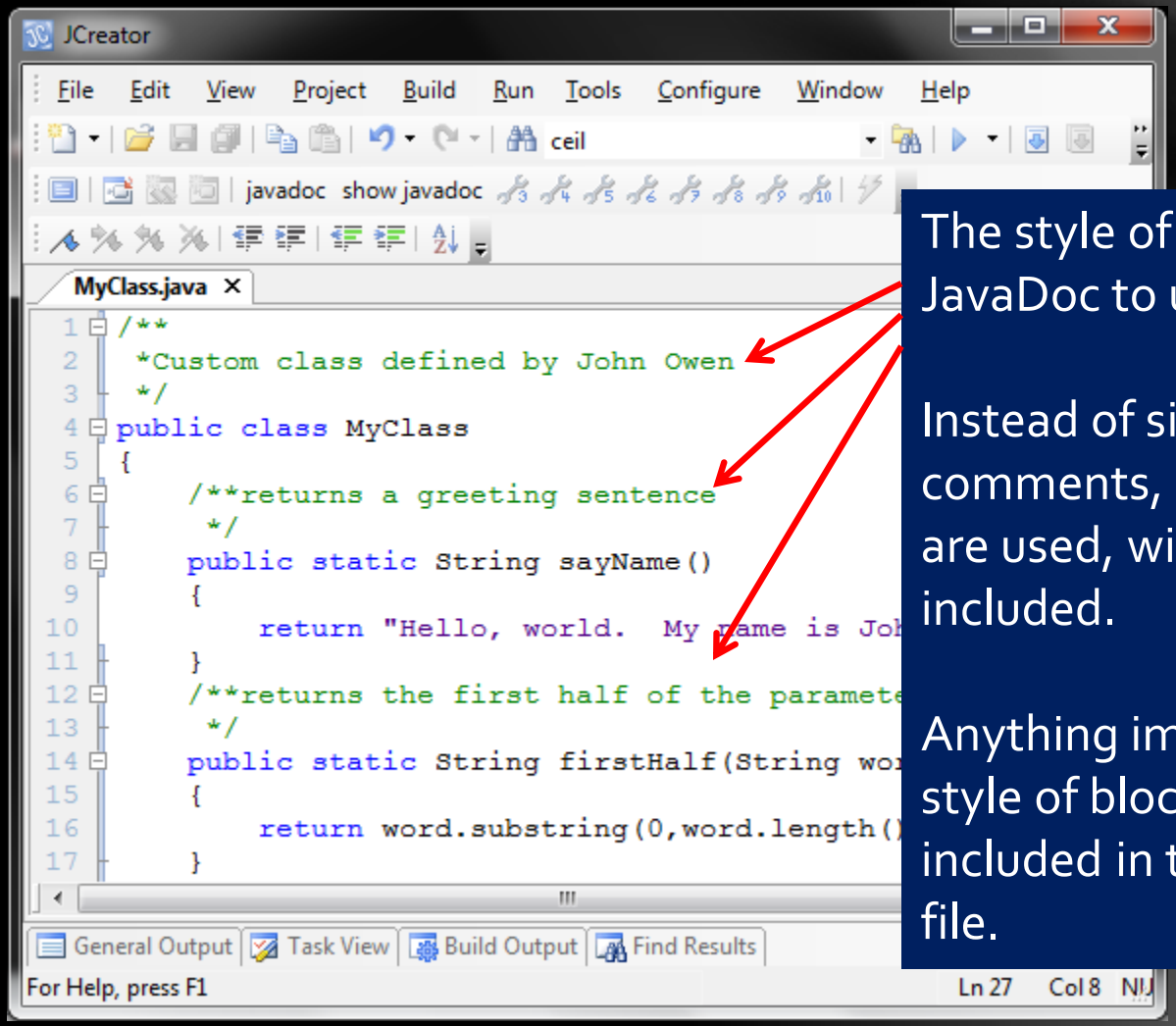
- We'll create two new tools in JCreator
  - javadoc - creates the help file
  - show javadoc - opens the help file

- The "javadoc" tool will create an html file from your class definitions, as long as you have included the correct documentation.

JavaDocs for JCreator

- The "show javadoc" tool will actually open that html file for you in a new browser window.
- **Oh, one more thing…**
- The style of documentation needs to be correct for JavaDoc to work.
- See the next slide for this…

JavaDocs for JCreator

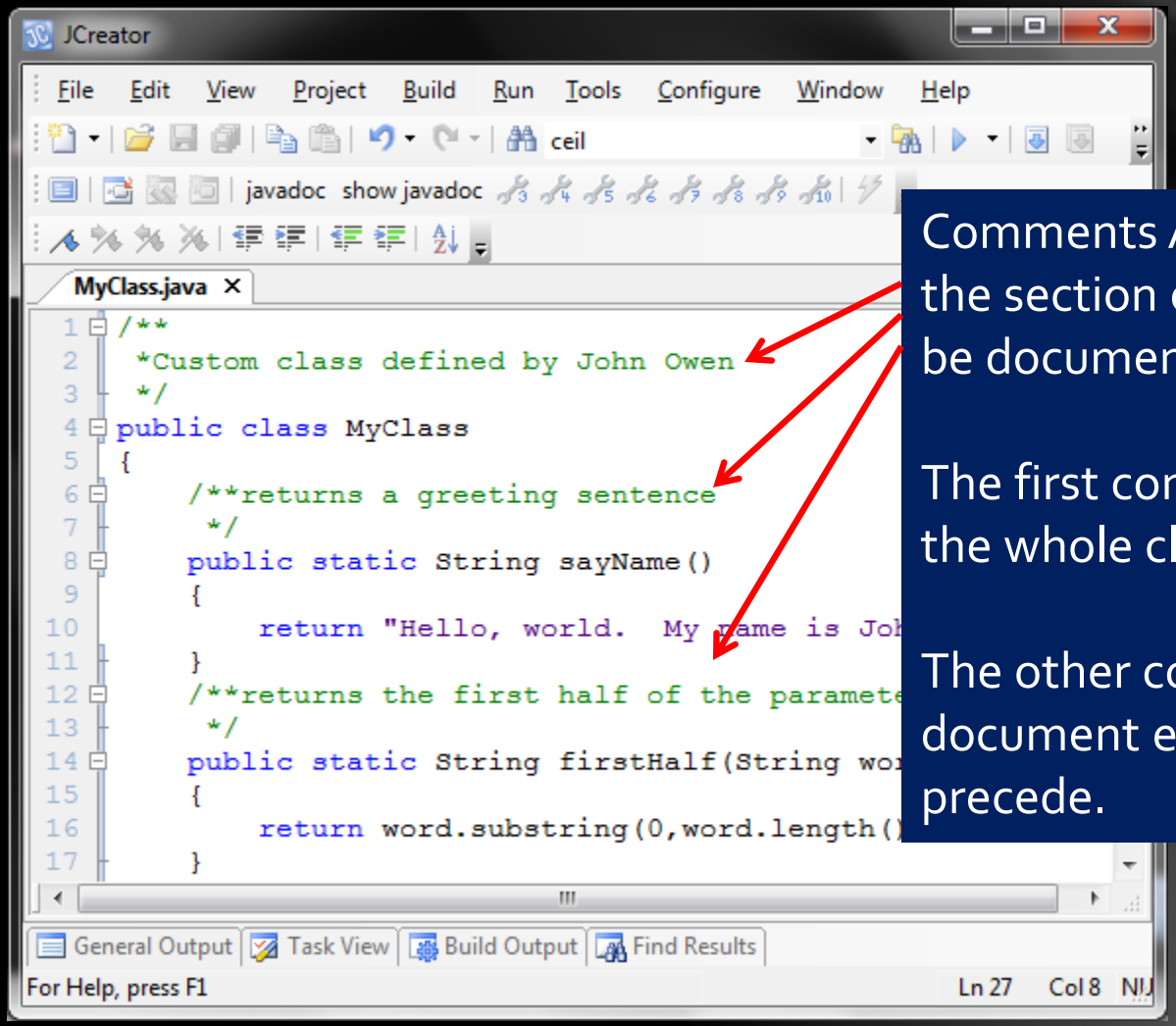


The style of comments for JavaDoc to use is as shown.

Instead of single line comments, block comments are used, with extra '*'s included.

Anything imbedded within this style of block comment will be included in the JavaDoc help file.

JavaDocs for JCreator

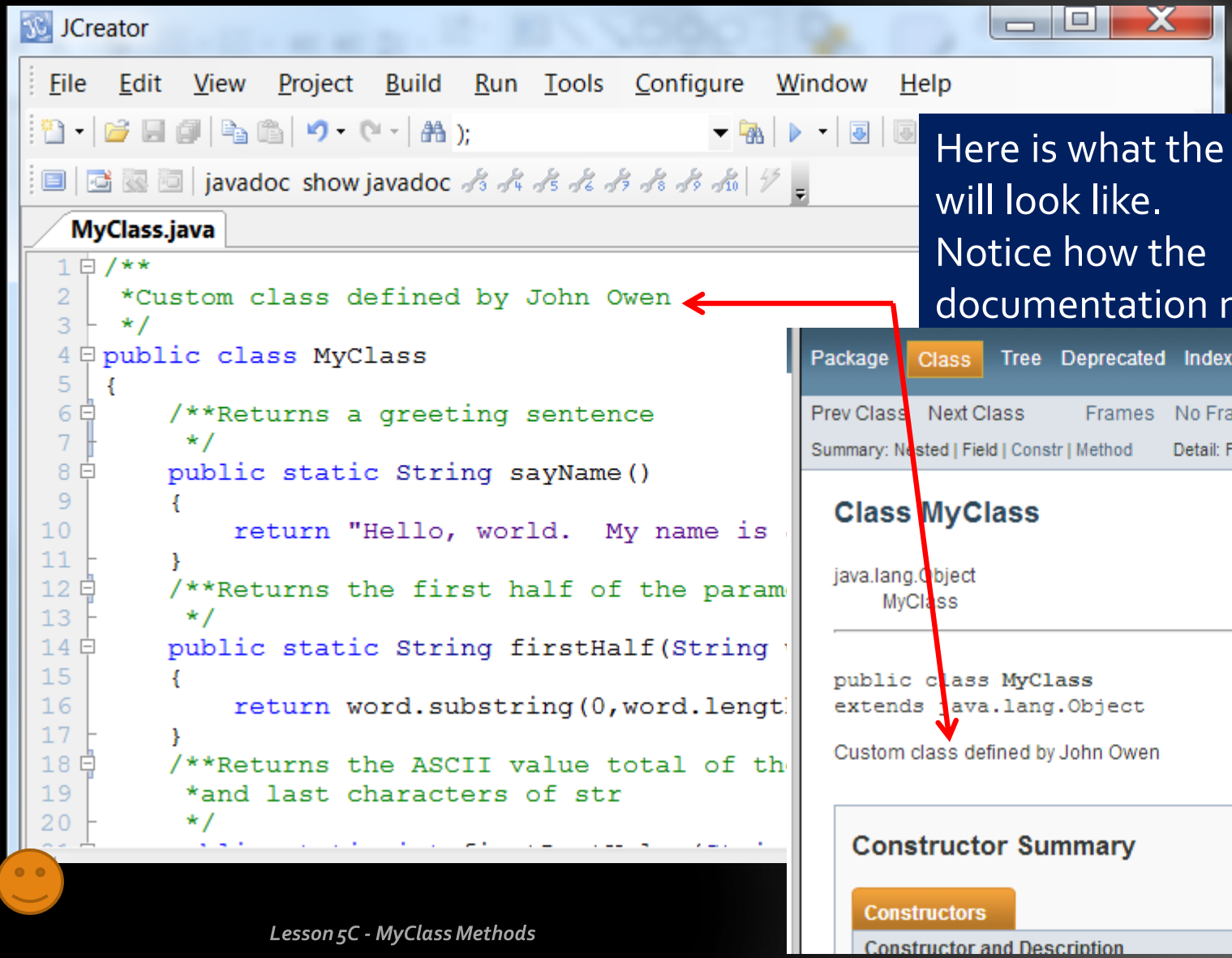


Comments ALWAYS precede the section of the program to be documented.

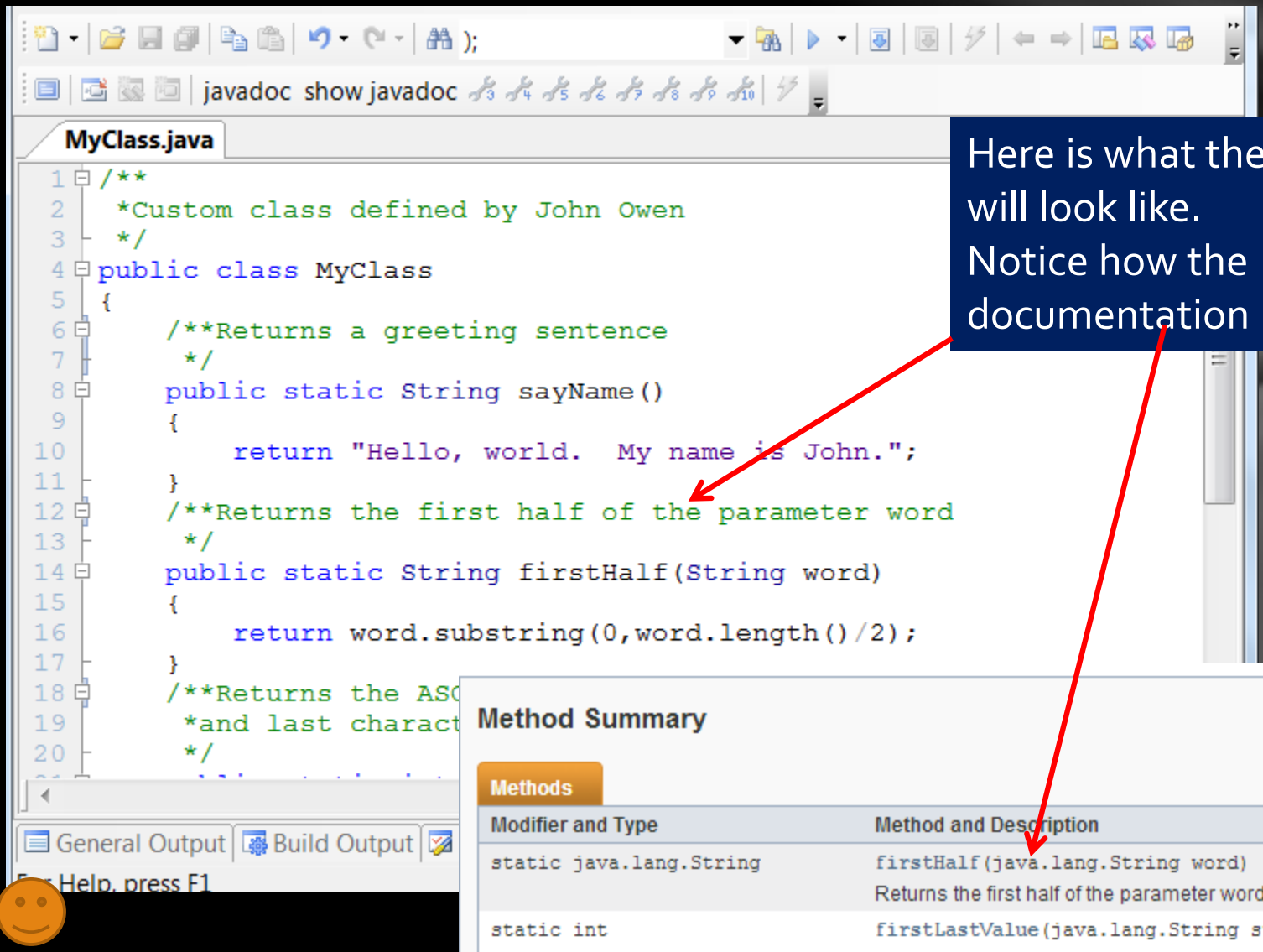
The first comment describes the whole class.

The other comments document each method they precede.

JavaDocs for JCreator



JavaDocs for JCreator

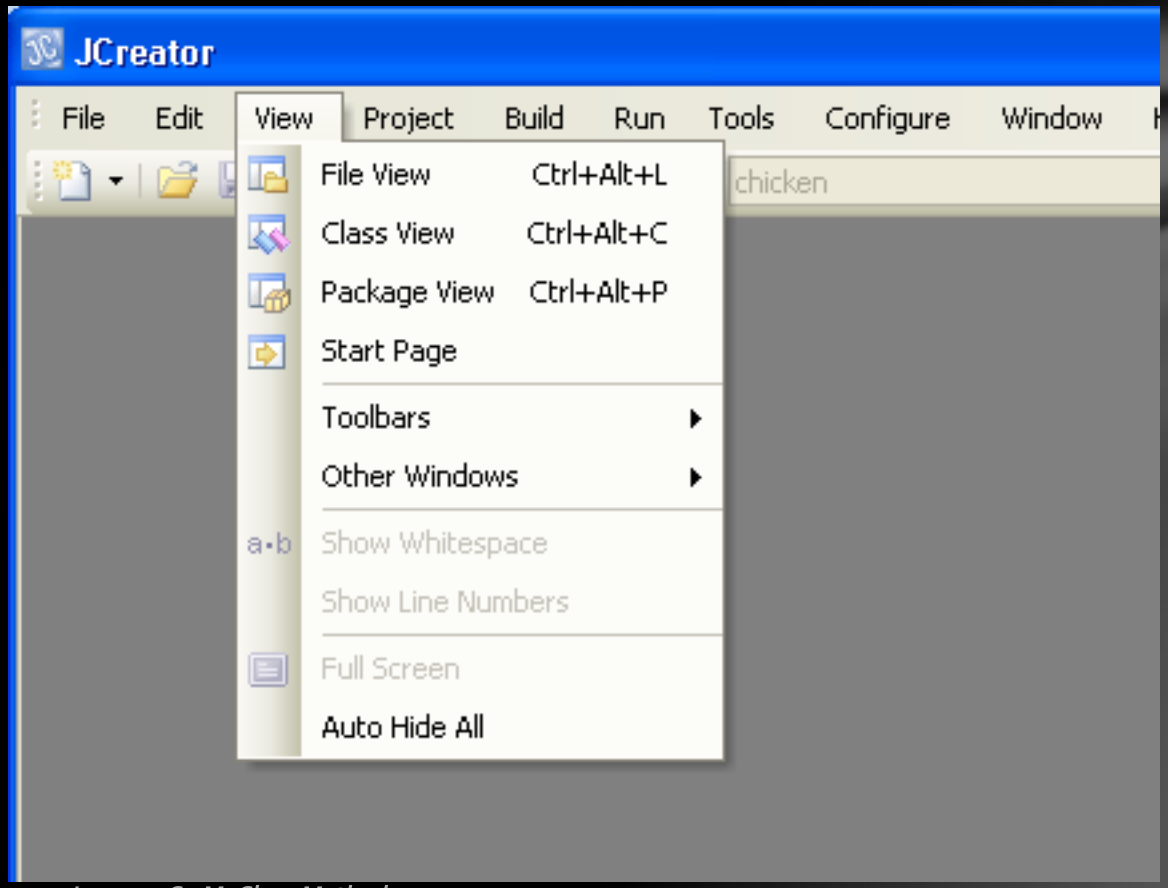


JavaDocs for JCreator

- Now let's create the JavaDoc tools…

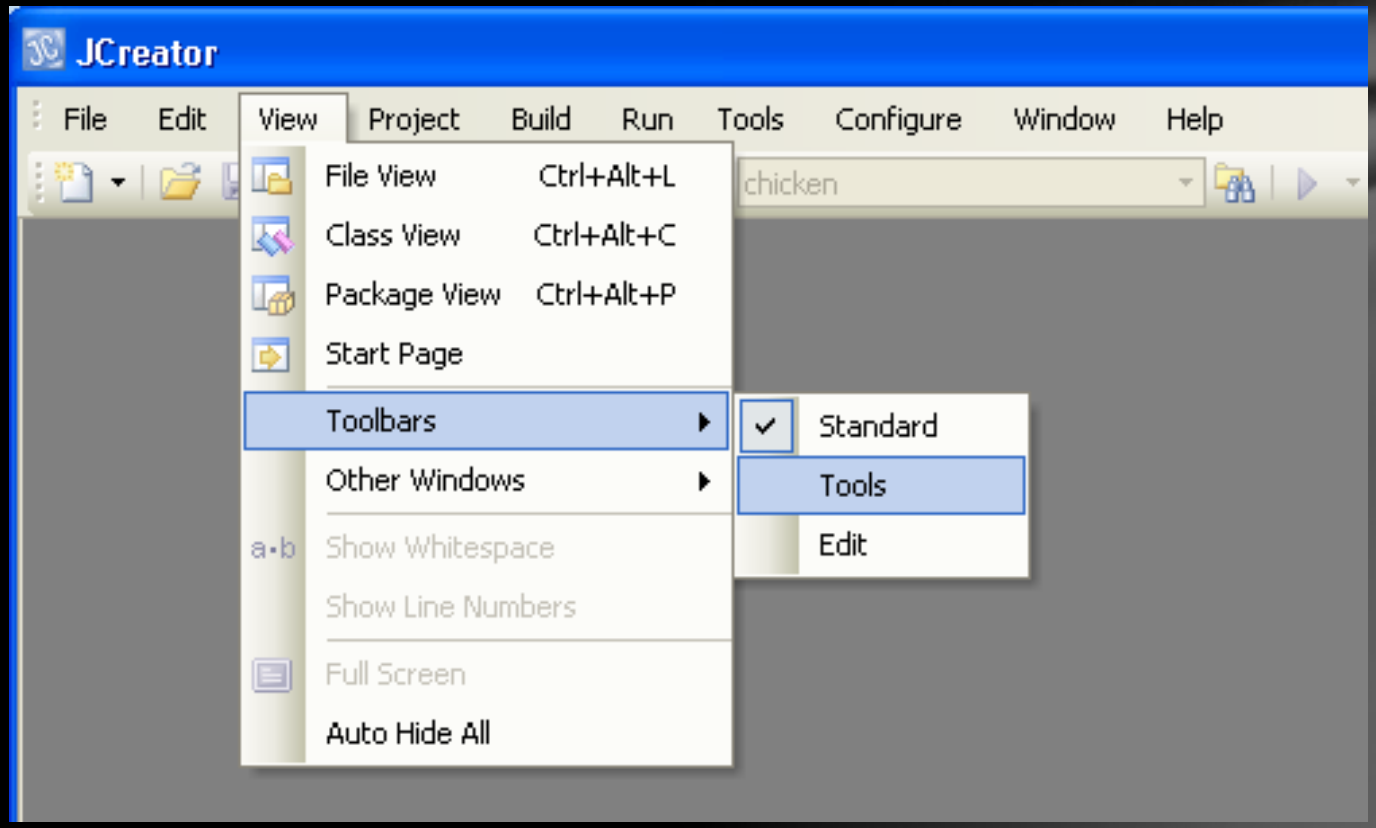
JavaDocs for JCreator

- Open Jcreator and click on the View tab at the top.



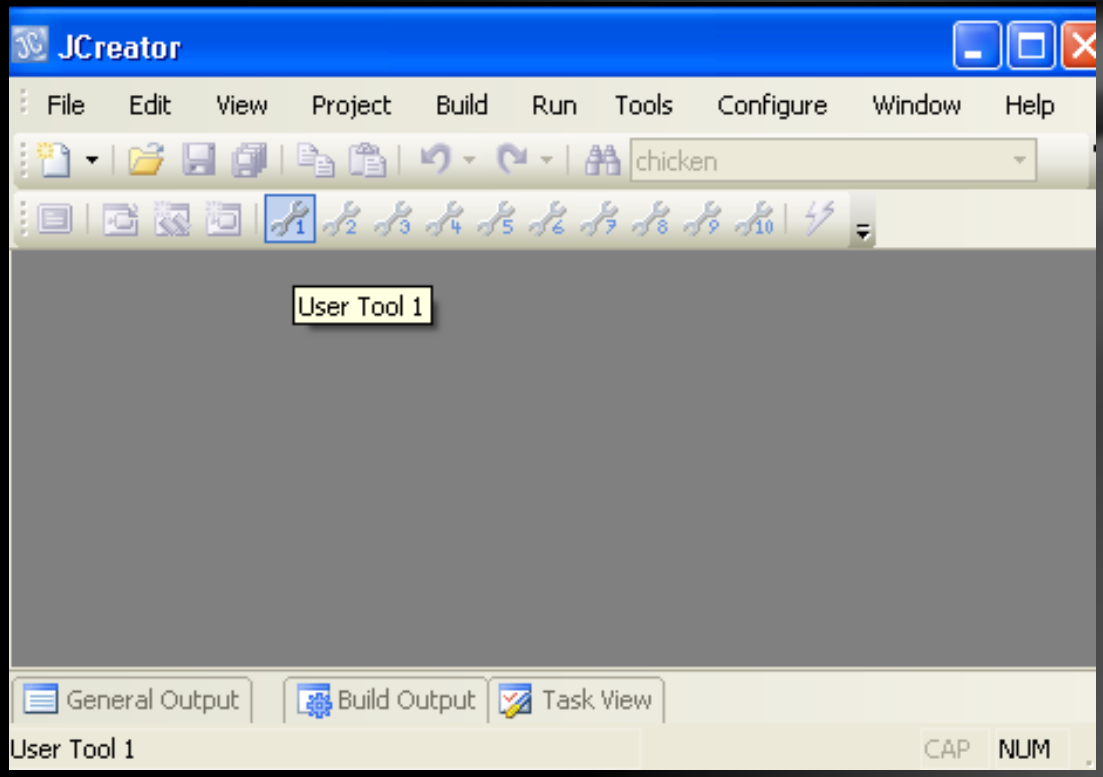
JavaDocs for JCreator

- Then click on the ToolBars black triangle and click on the Tools button.



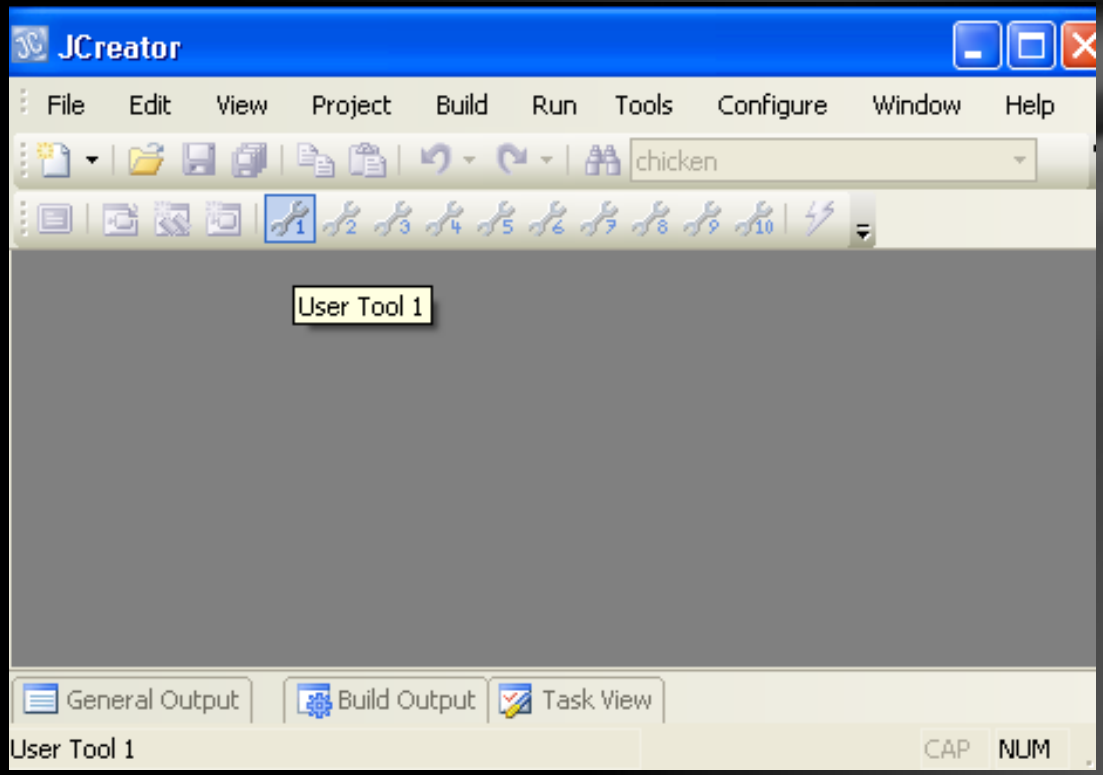
JavaDocs for JCreator

- You should now see a row of Tools that look like wrenches. Hover over the first one to see its name – User Tool 1



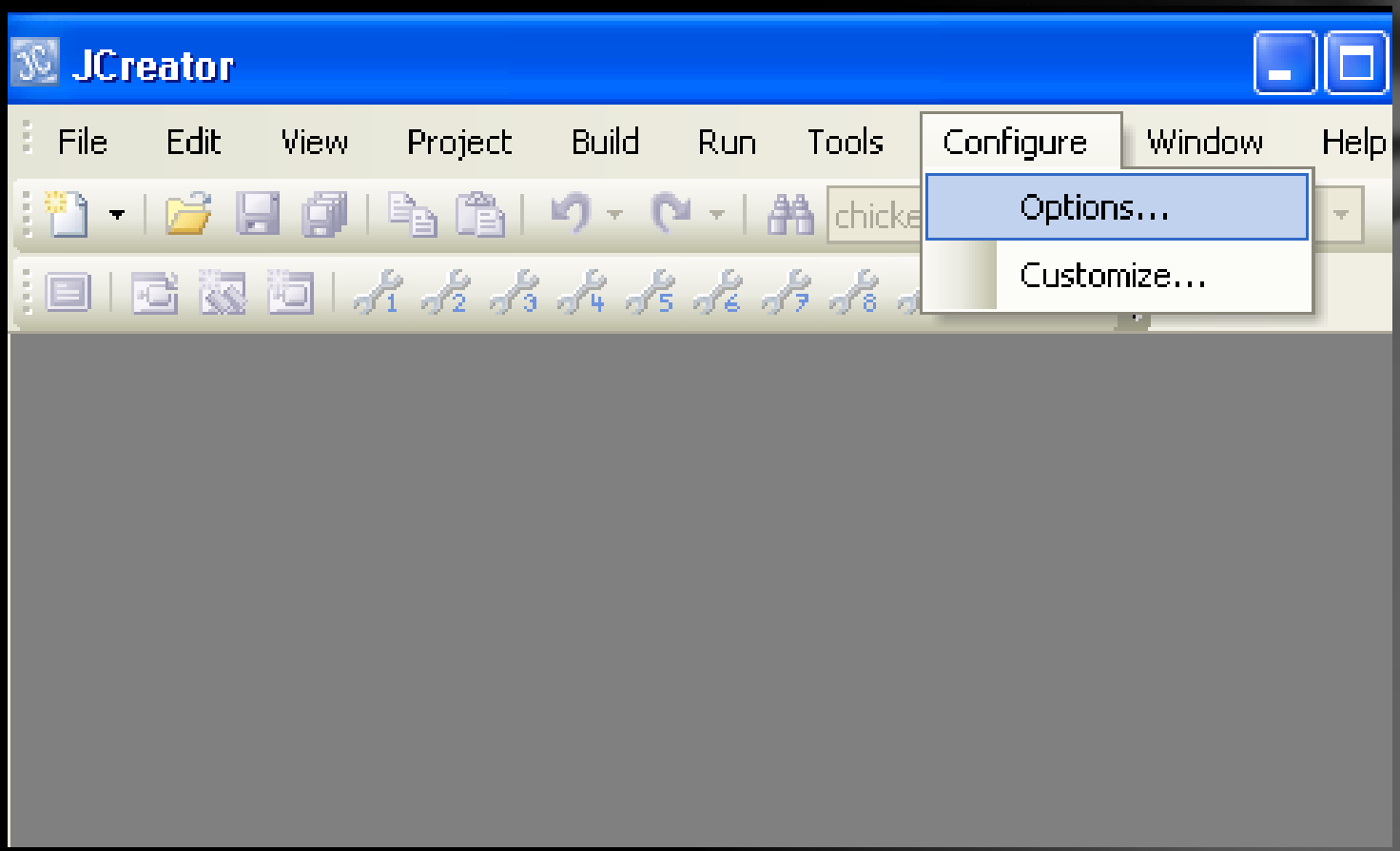
JavaDocs for JCreator

- We will now redefine this tool as the "javadoc" tool. Here's how…



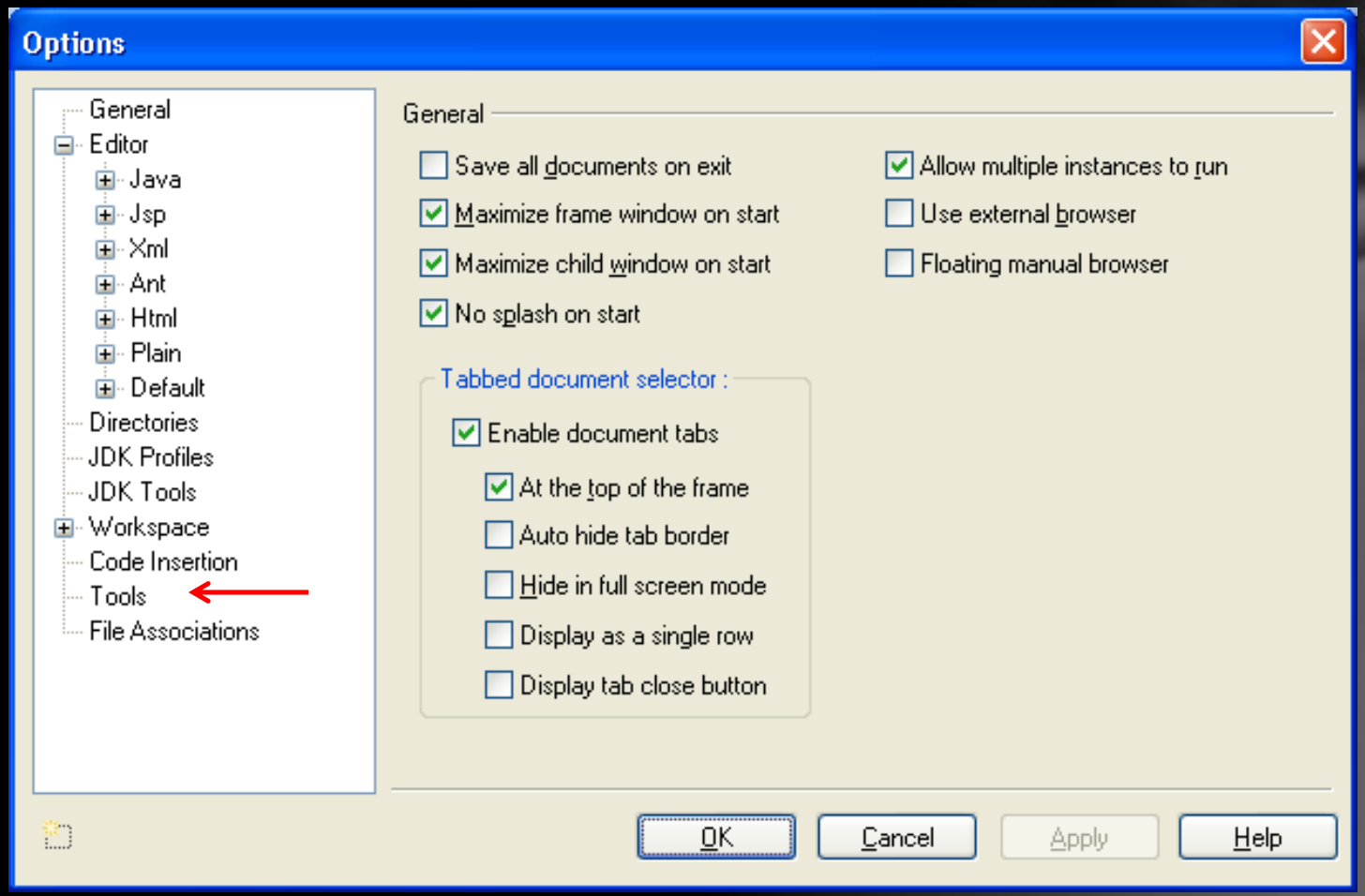
JavaDocs for JCreator

- Click on Configure, then Options…



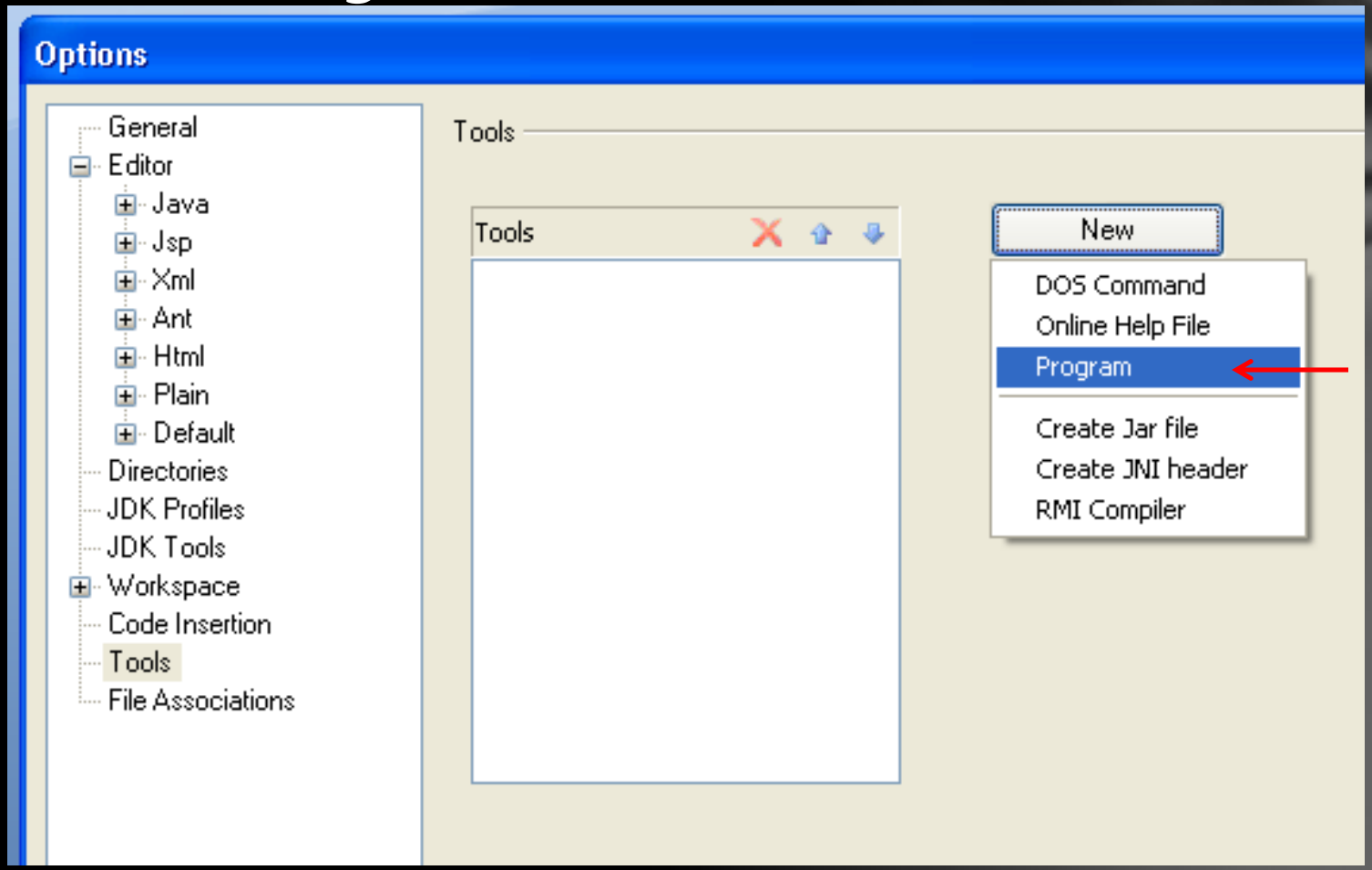
JavaDocs for JCreator

- From here, select Tools…



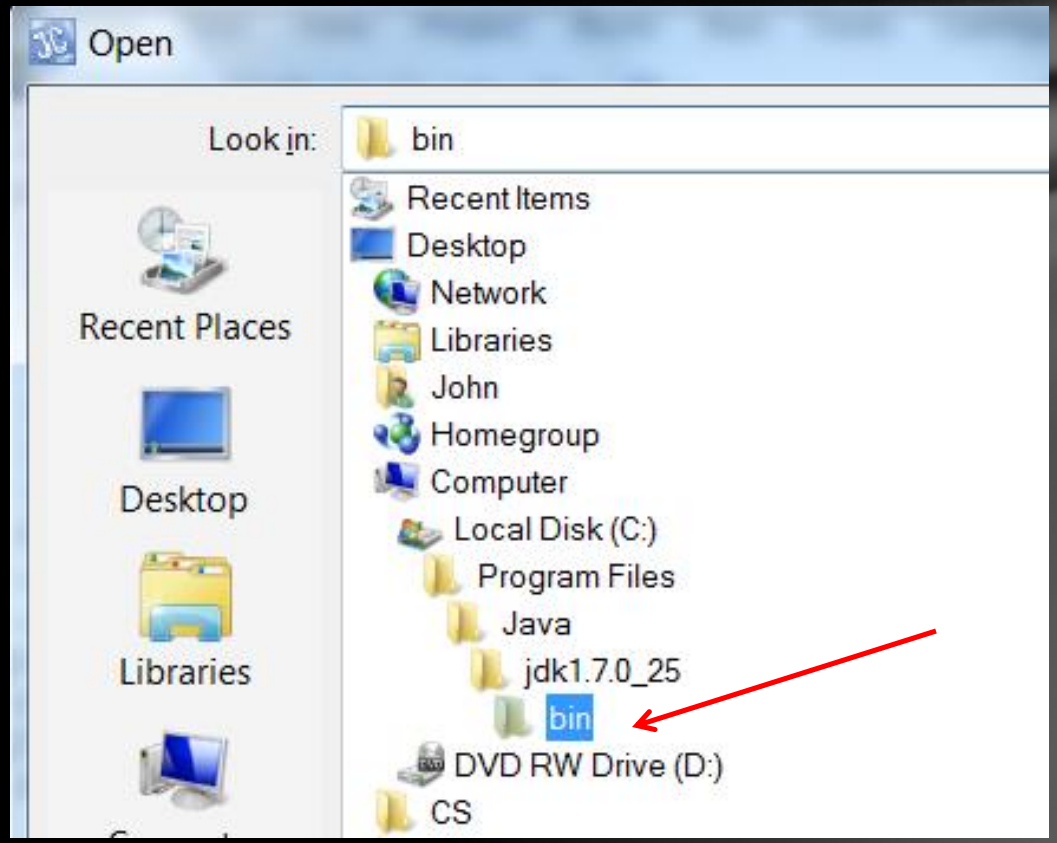
JavaDocs for JCreator

- …then Program…



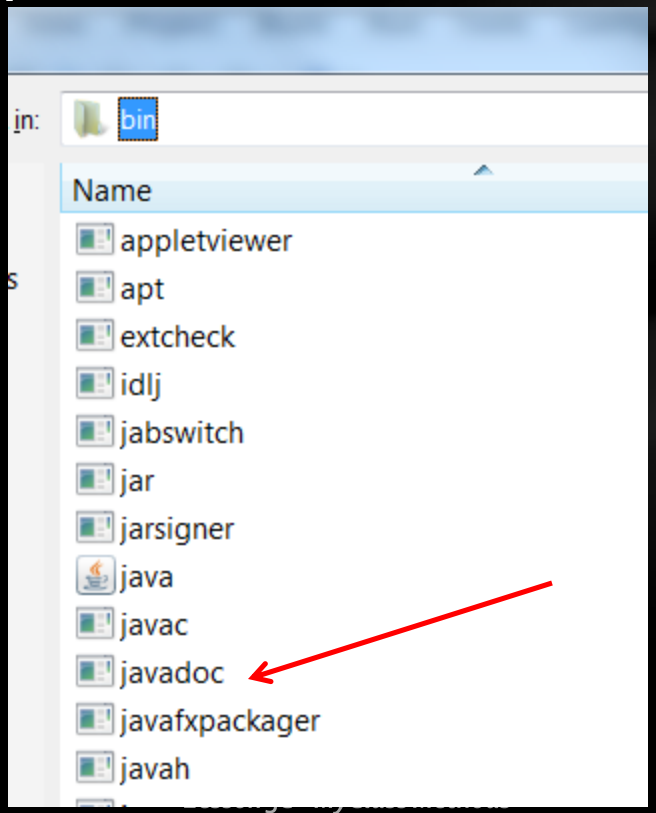
JavaDocs for JCreator

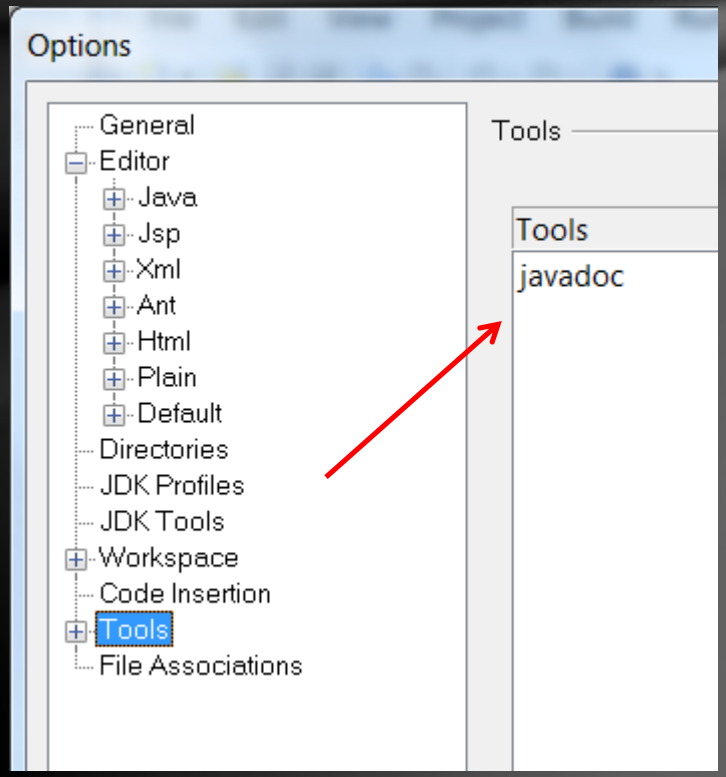
- …then navigate to wherever Java "lives" on your computer, specifically to the **bin** folder…



JavaDocs for JCreator

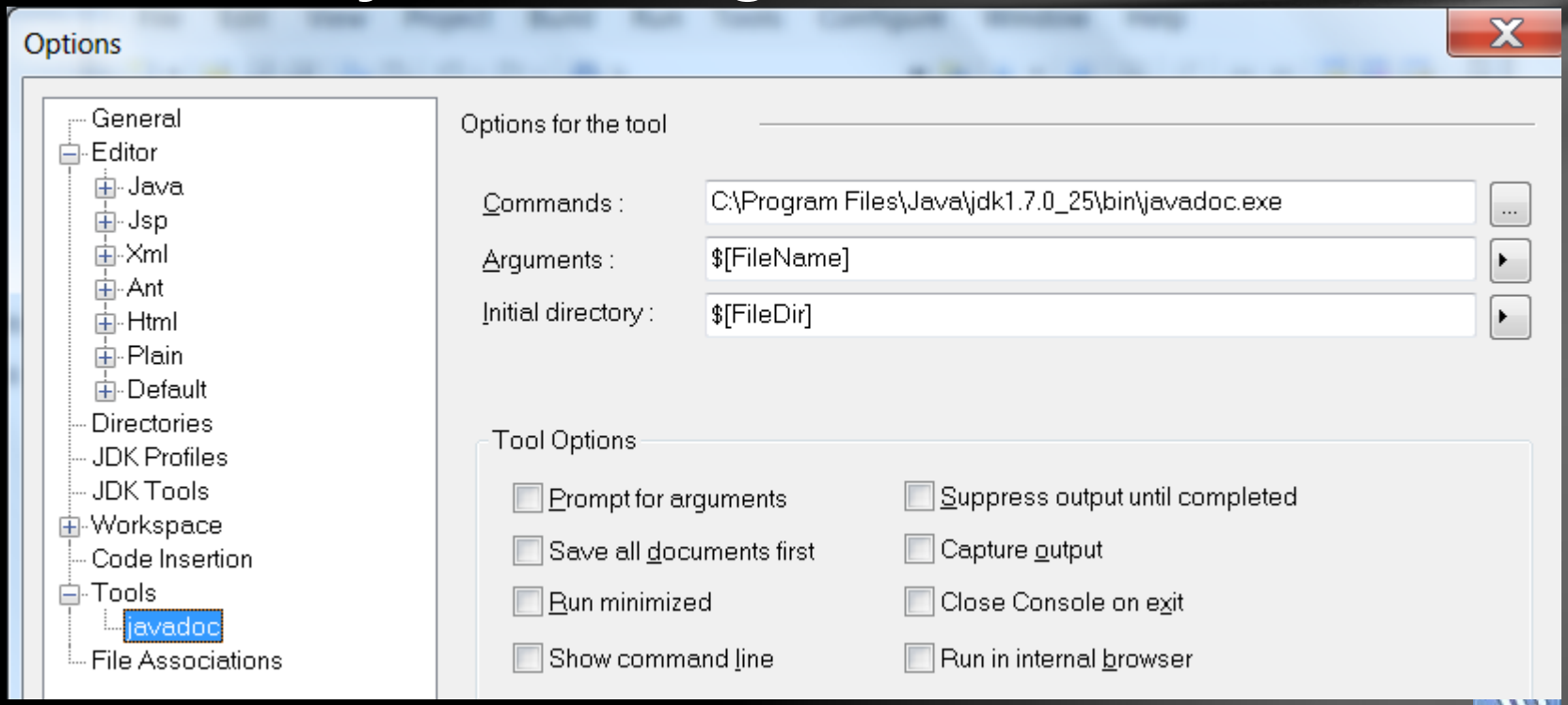
- Inside the **bin** folder is a file called javadoc.exe…double click on it and you'll see a new tool appear in JCreator





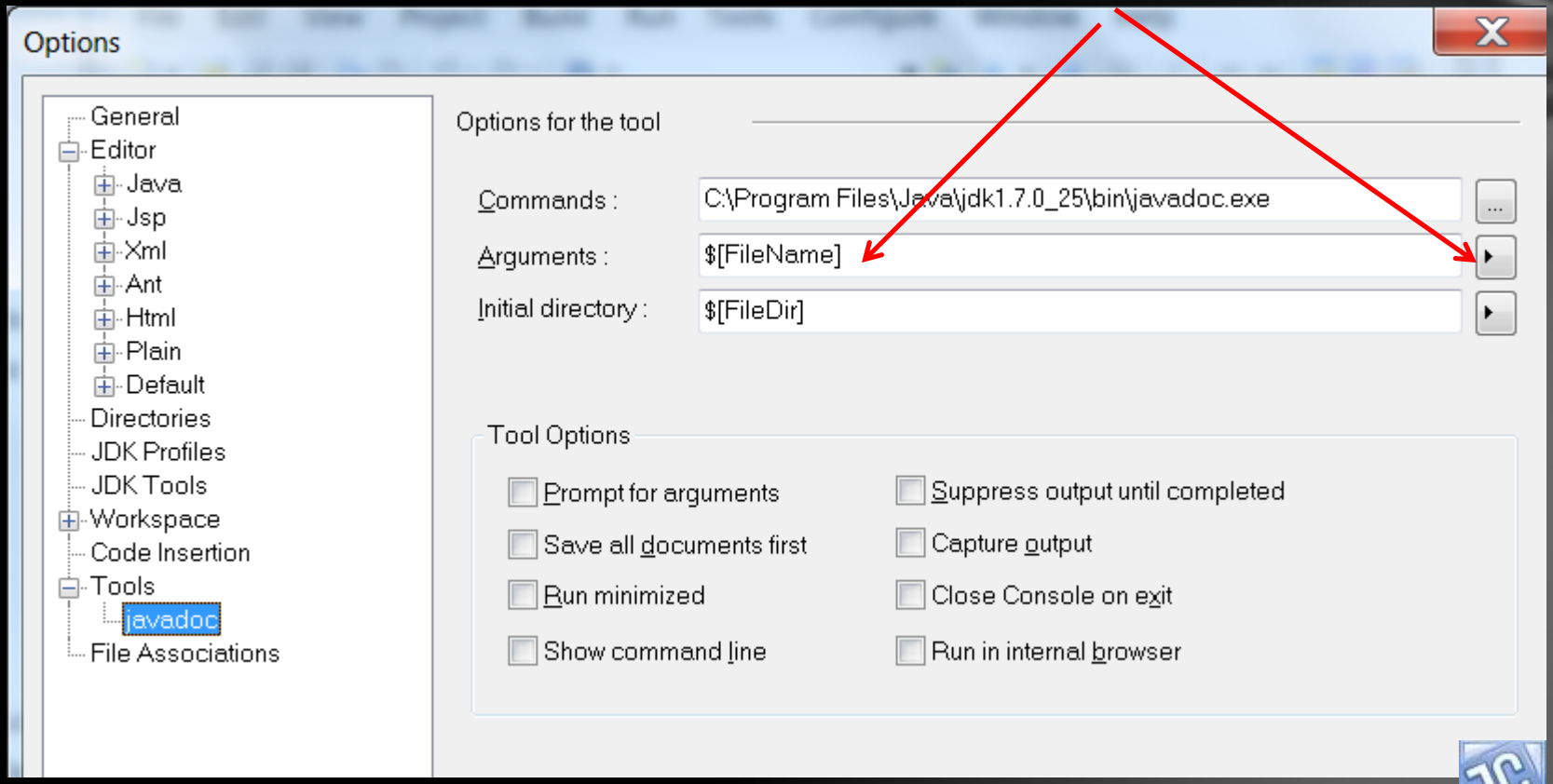
JavaDocs for JCreator

- To complete the configuration of this tool, click on Configure, Options, Tools, and then javadoc to get to this window.



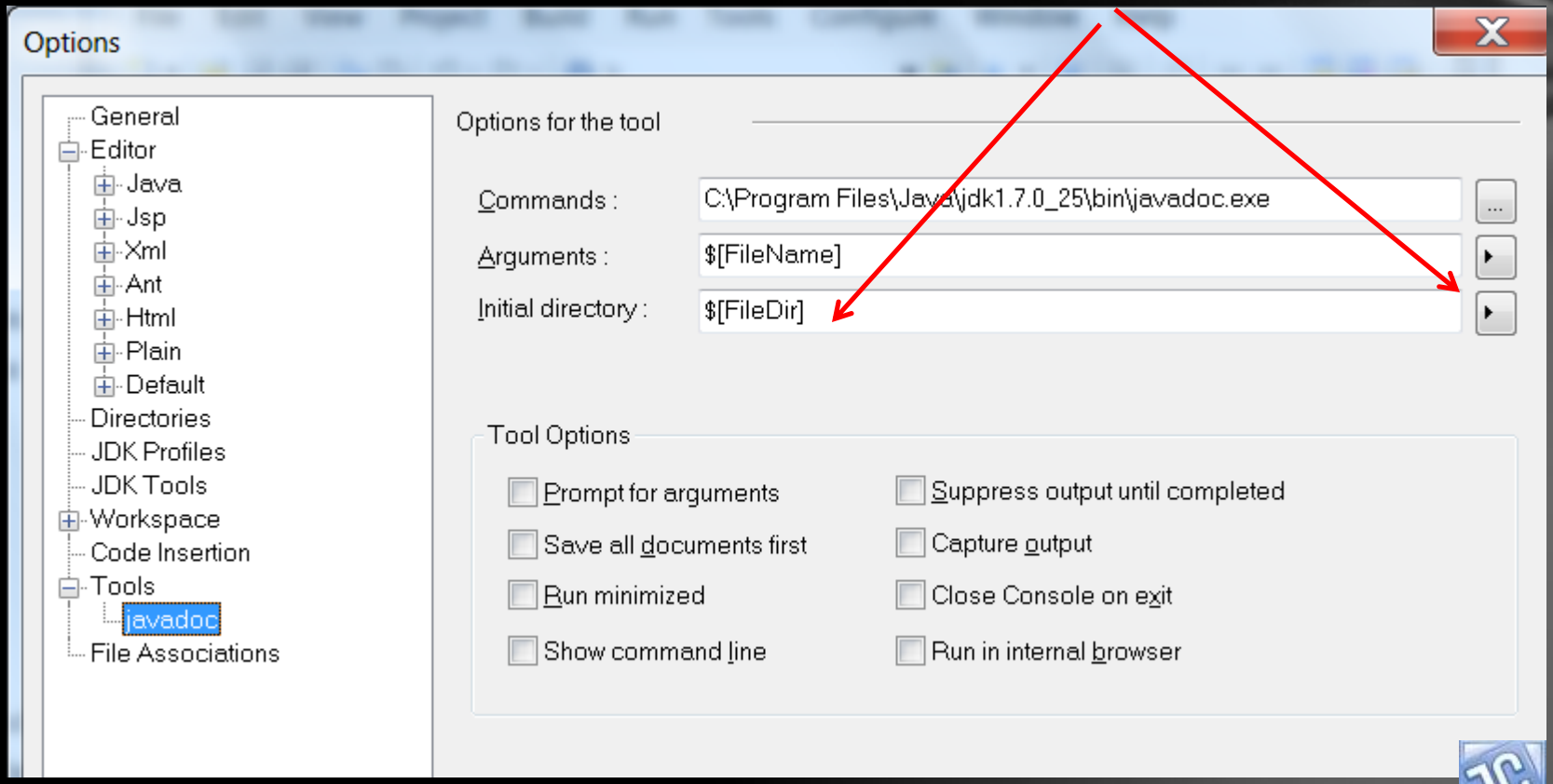
JavaDocs for JCreator

- For the Arguments window, click on the arrow button and select File Name

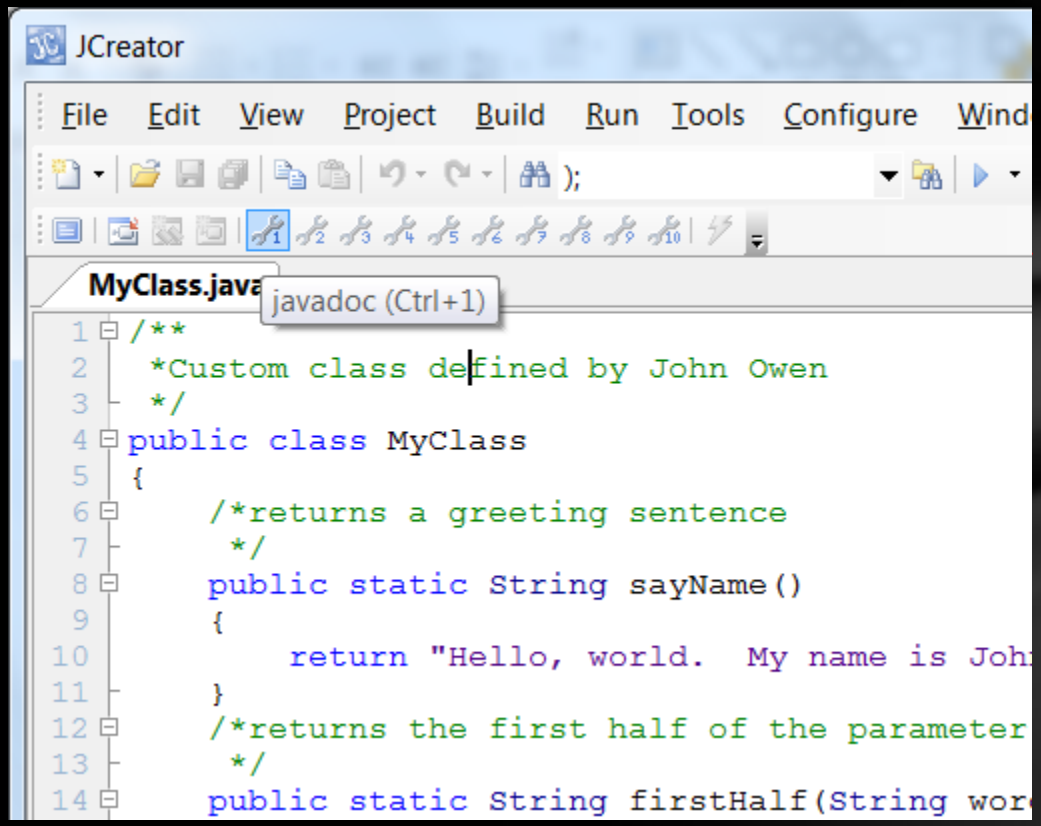


JavaDocs for JCreator

- For the Initial Directory, click on the arrow button and select File Directory



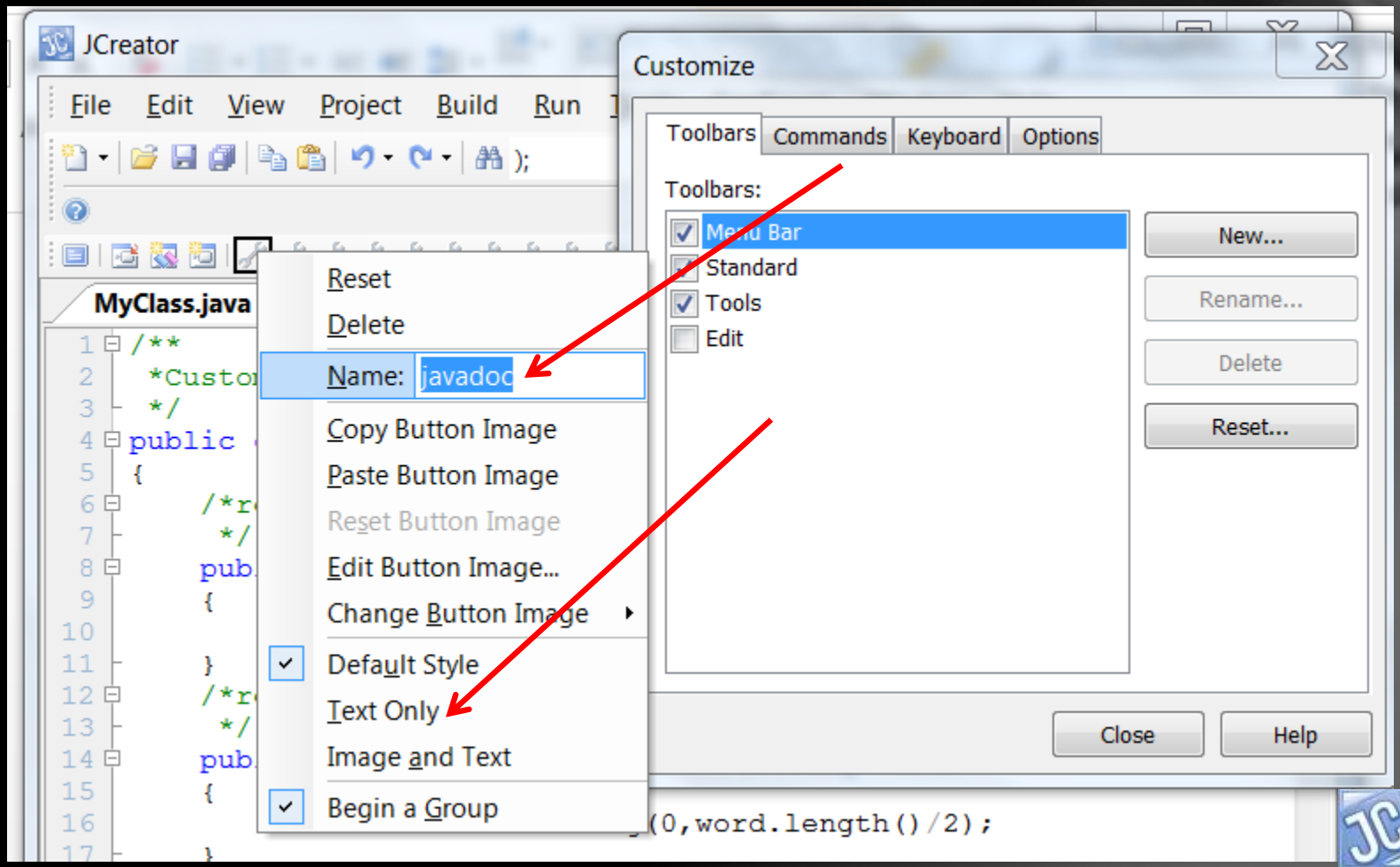
JavaDocs for JCreator



Now when you hover over the first wrench, it says "javadoc" as the name of the tool.

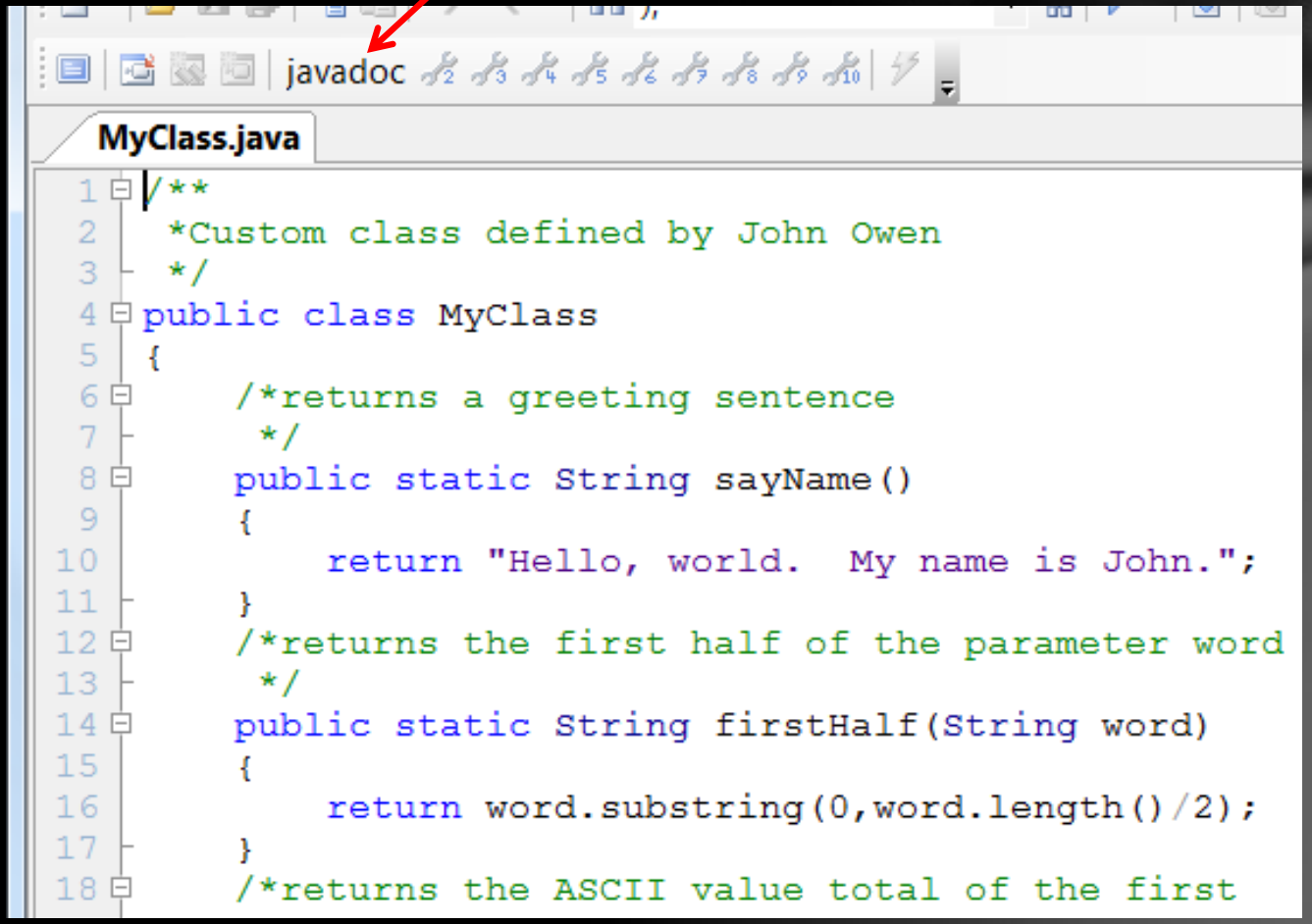
JavaDocs

To change the tool appearance from a wrench to the label "javadoc", right click once, click Customize, move that window to the side and ignore it, then right click again, change the name to "javadoc", and then click Text Only



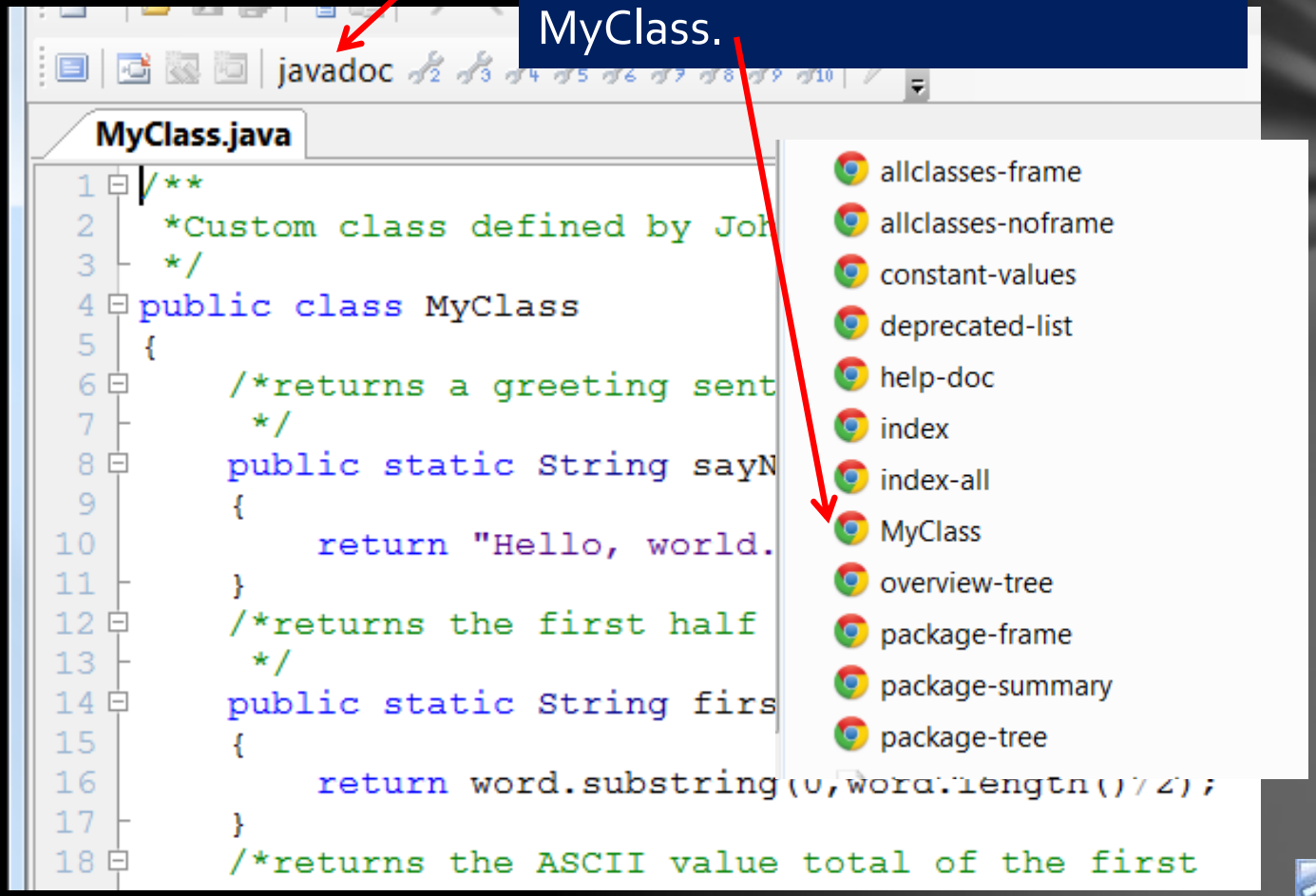
JavaDocs for JCreator

Now you can see the tool name as you changed it.



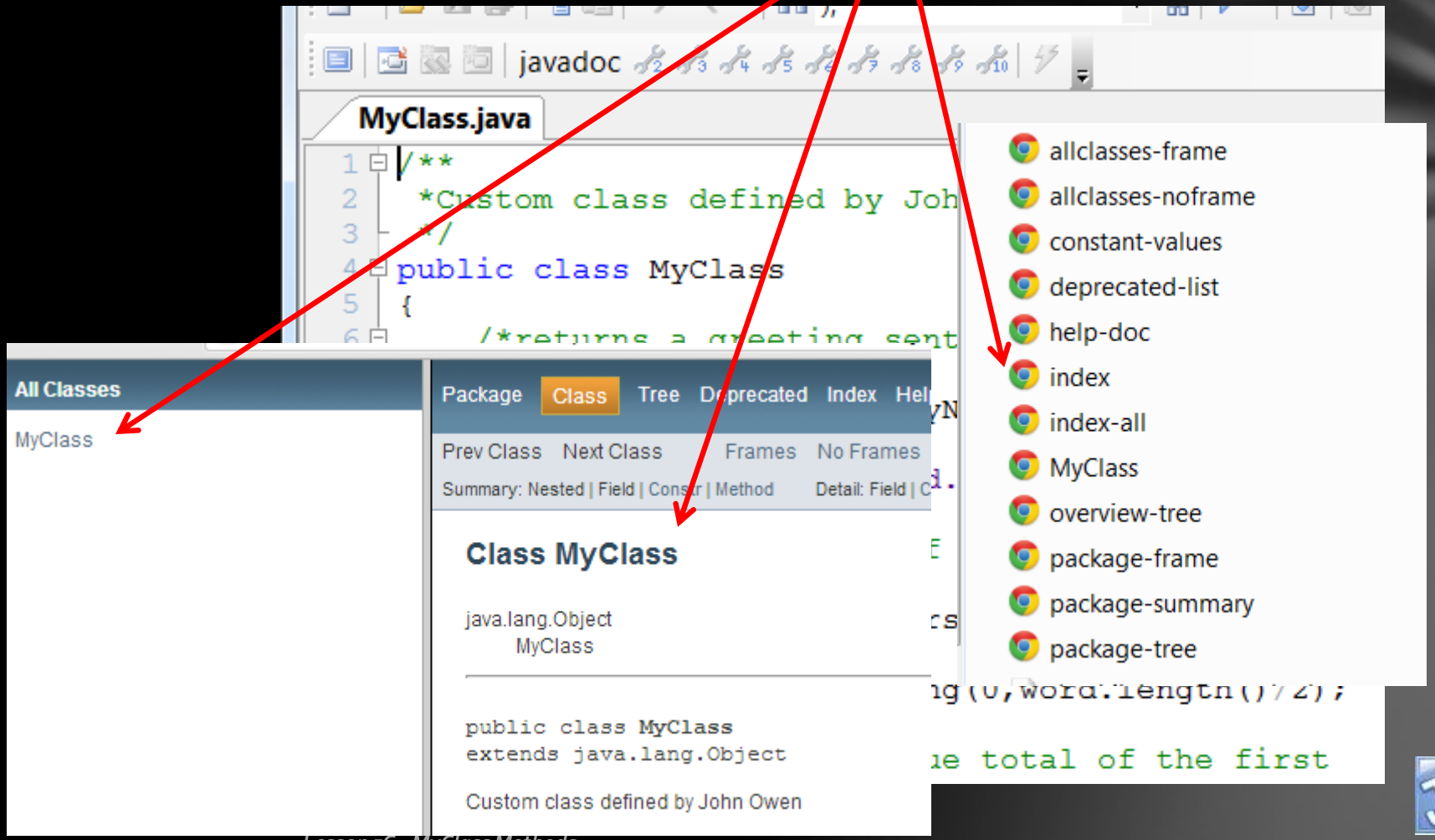
JavaDocs for JCreator

When you click on it, a series of html files will be created, as shown below, including one for MyClass.



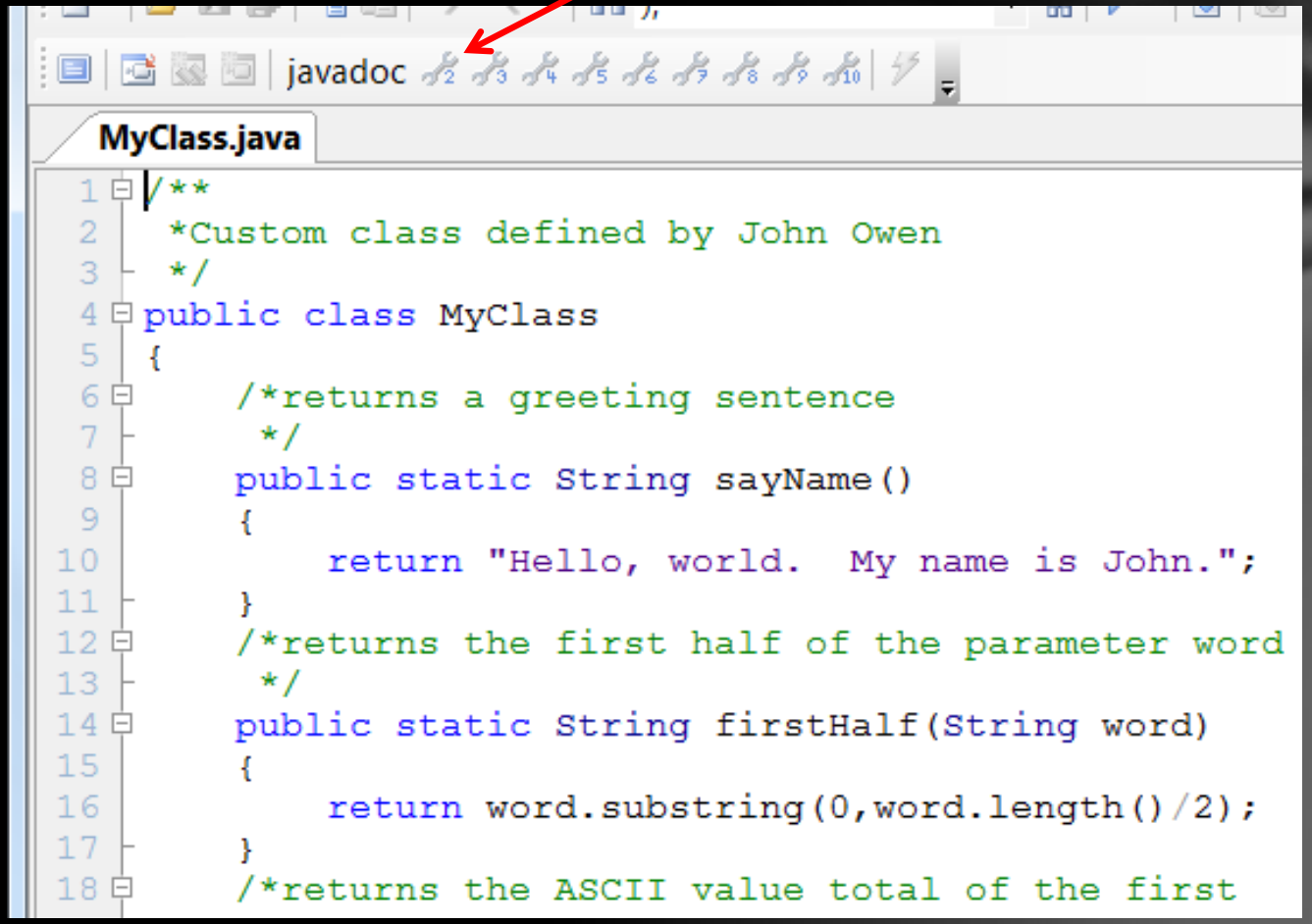
JavaDocs for JCrea

Open the one called "index" for the main page to see the Java Documentation for this file.

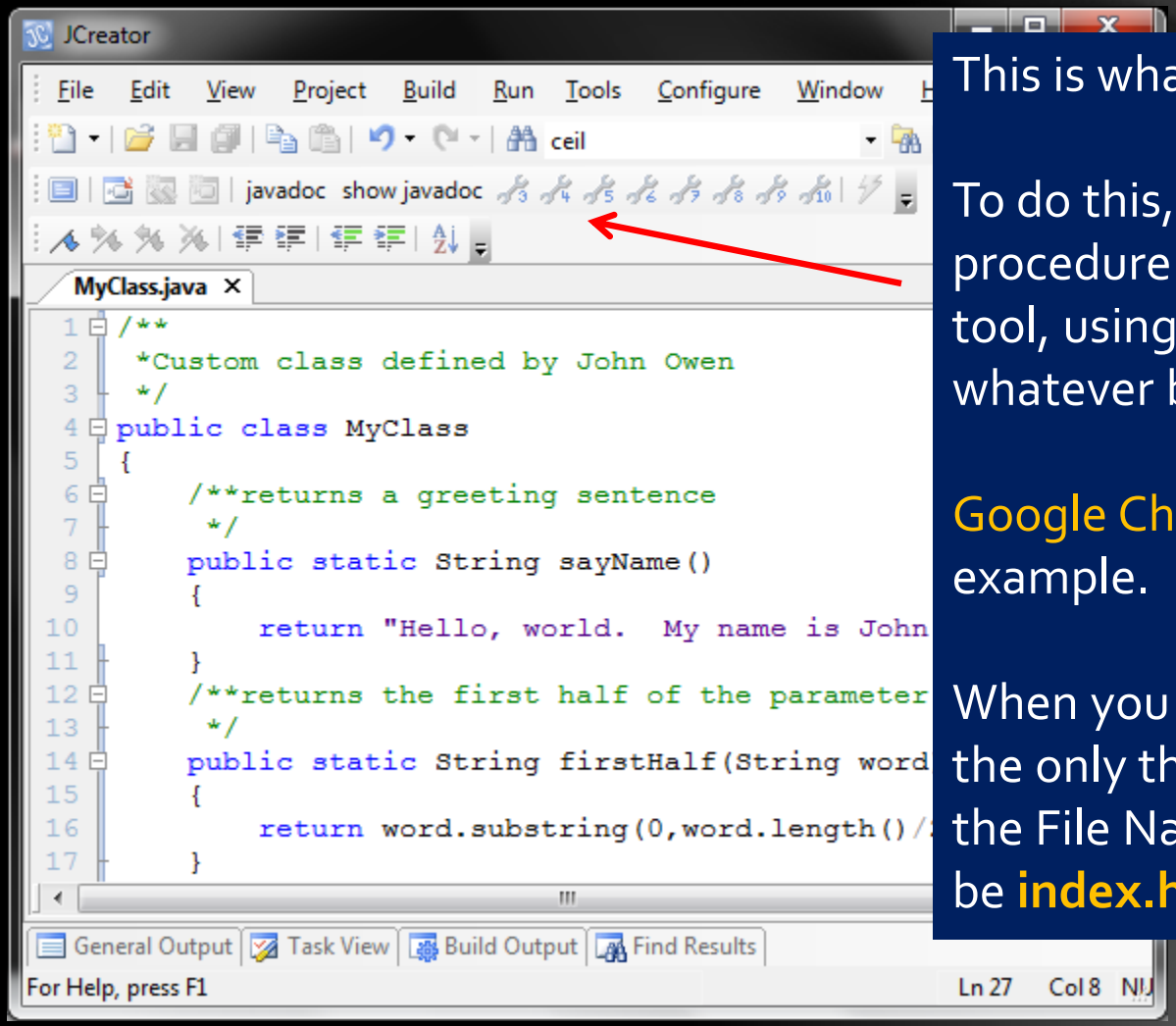


JavaDocs fo

Now we'll create a second tool that can be clicked to automatically bring up the JavaDoc for the current file. We'll call it show JavaDoc



JavaDocs for JCreator



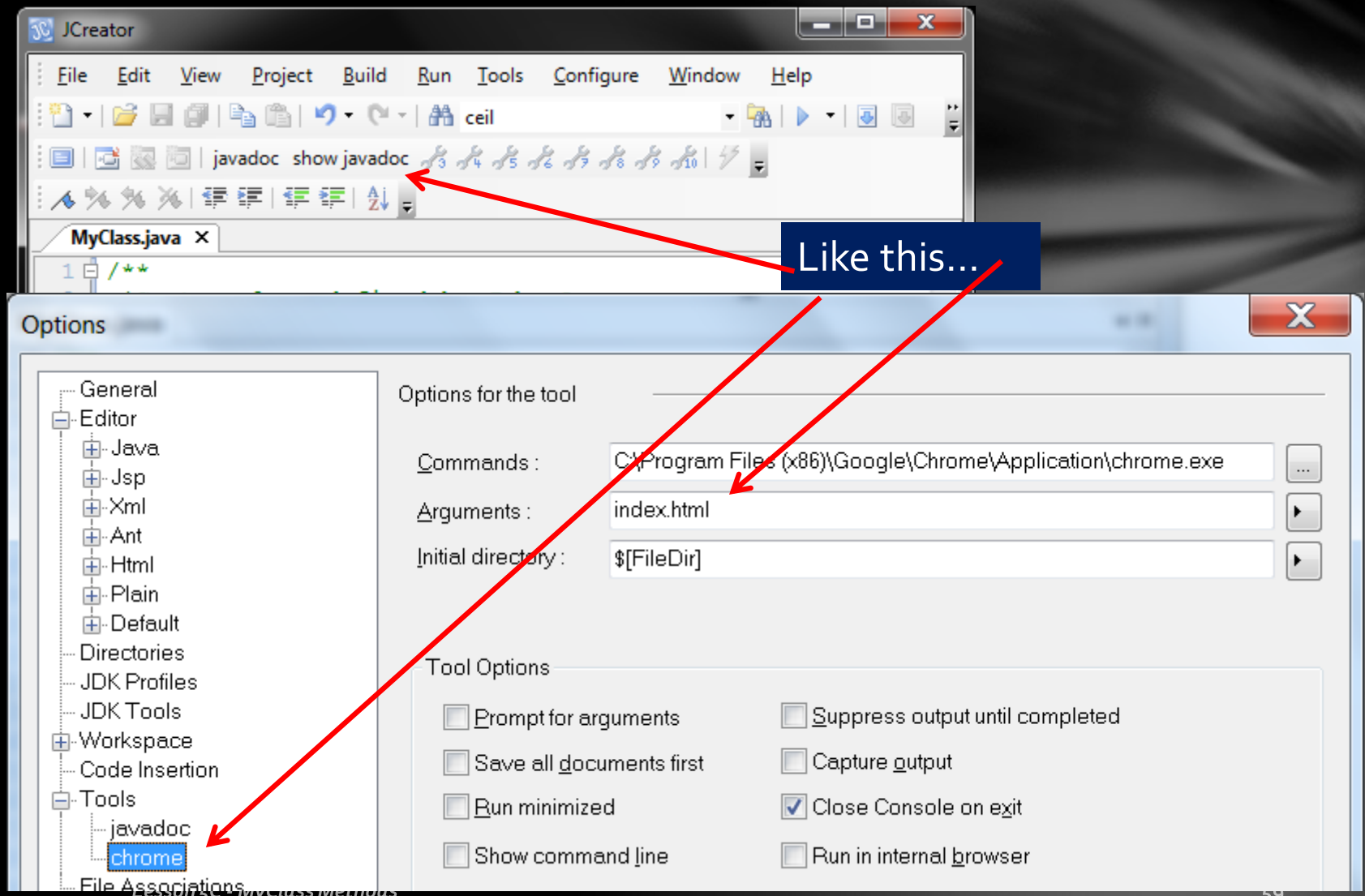
This is what it should look like.

To do this, follow the same procedure you did for the first tool, using the program for whatever browser you prefer.

Google Chrome is used for this example.

When you get to the Options, the only thing different will be the File Name, which should be **index.html**

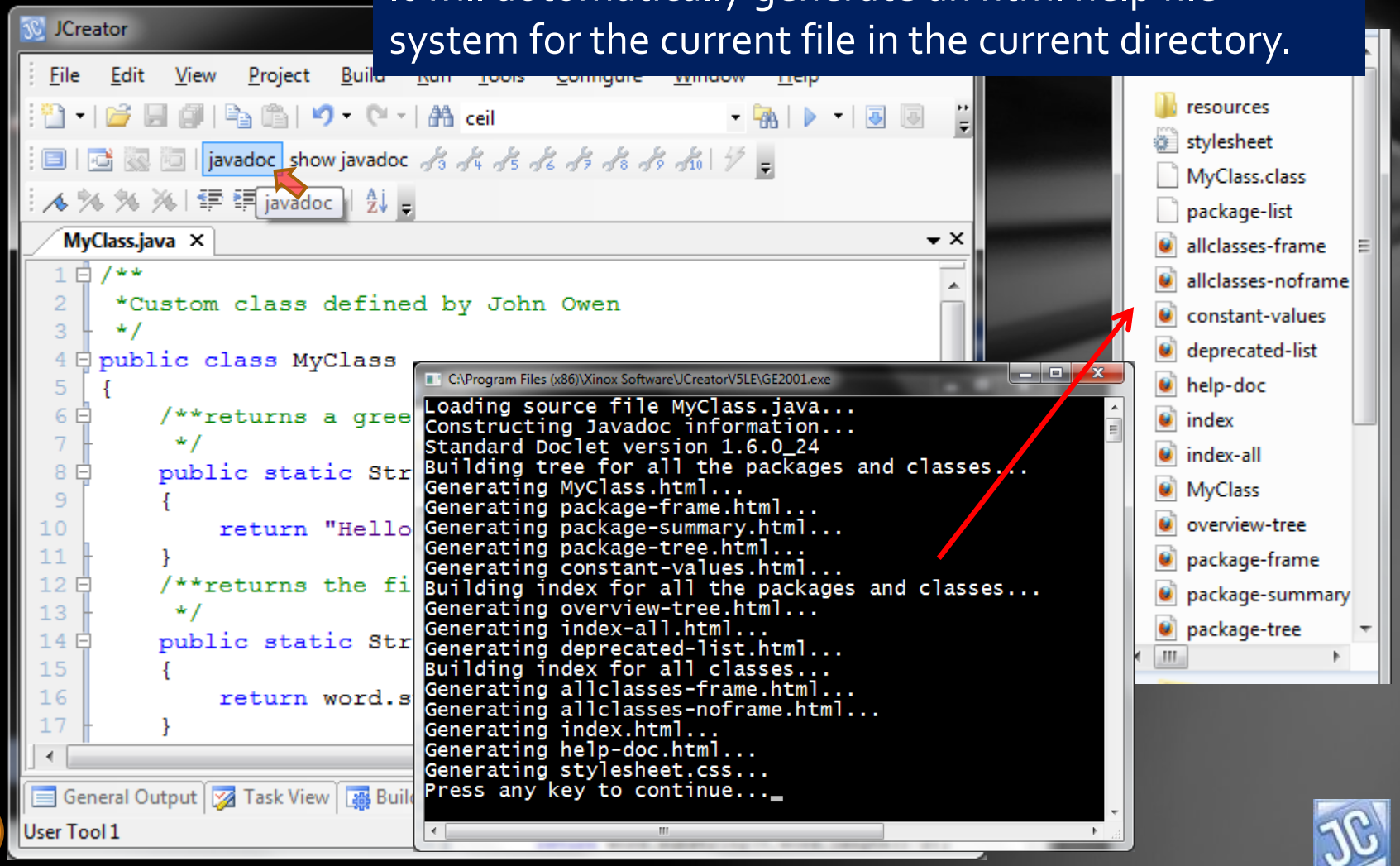
JavaDocs for JCreator



JavaDocs fo

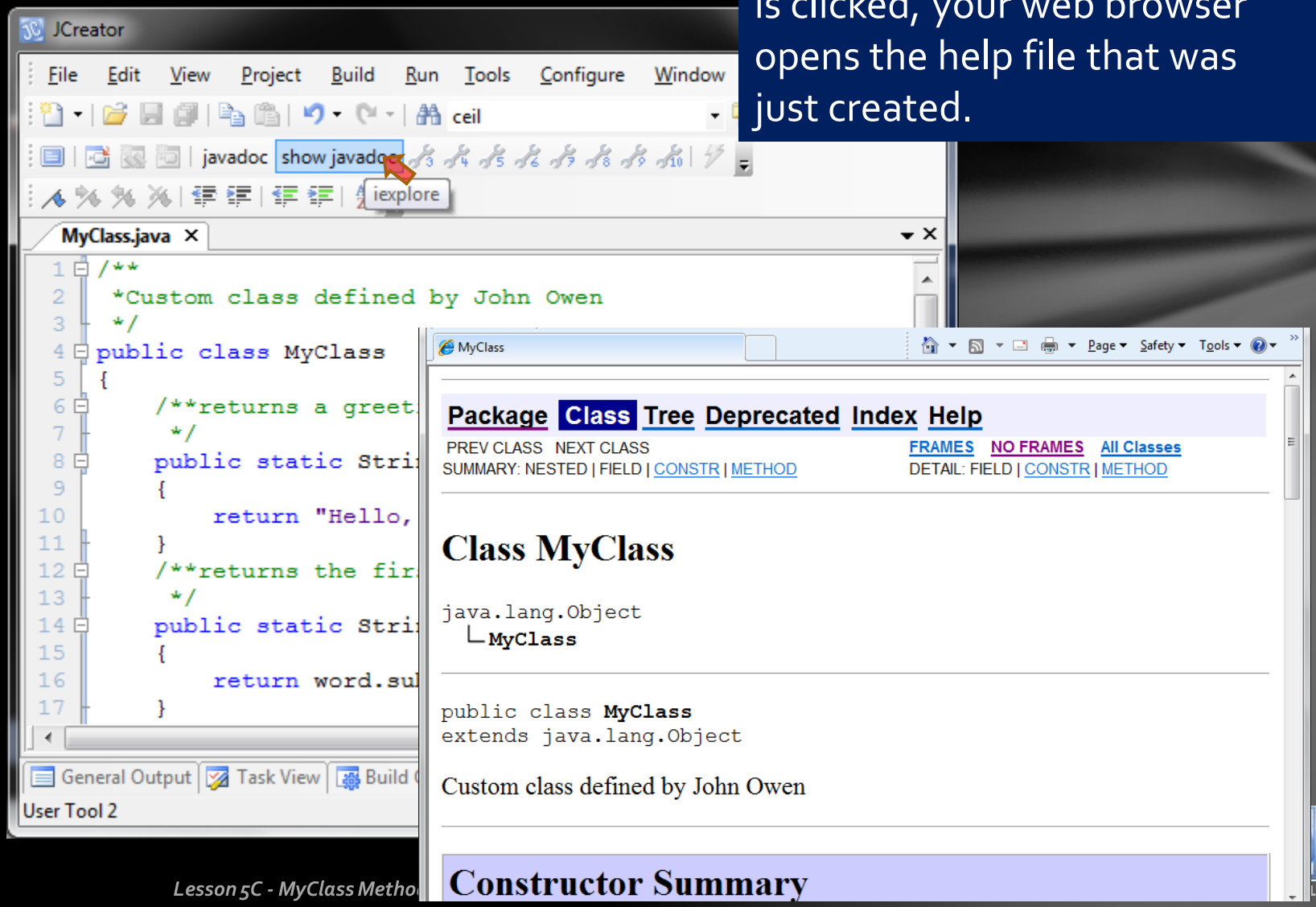
To run the JavaDoc tool, simply click on it in the JCreator window.

It will automatically generate an html help file system for the current file in the current directory.



JavaDocs for JCreator

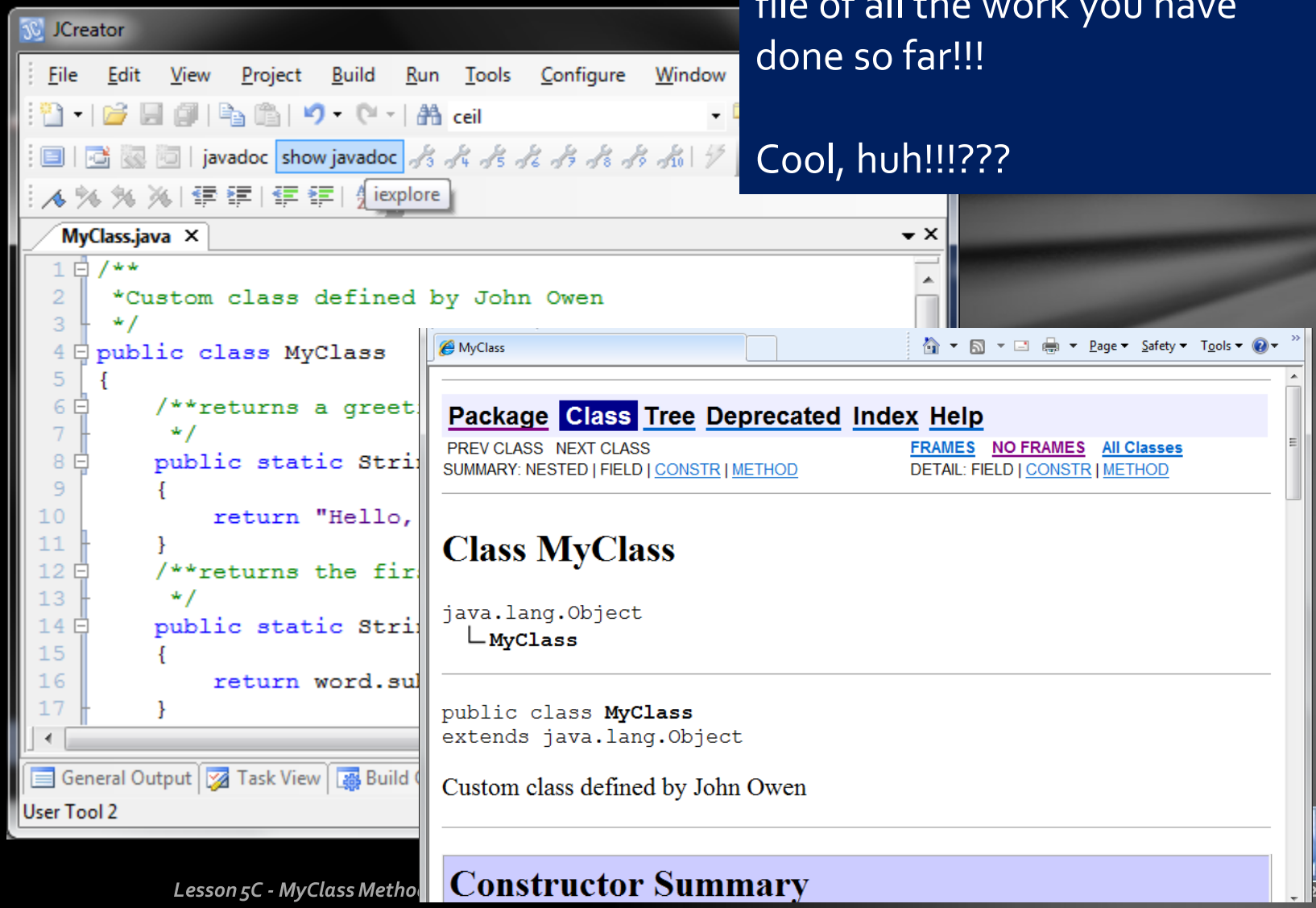
When the "show javadoc" tool is clicked, your web browser opens the help file that was just created.



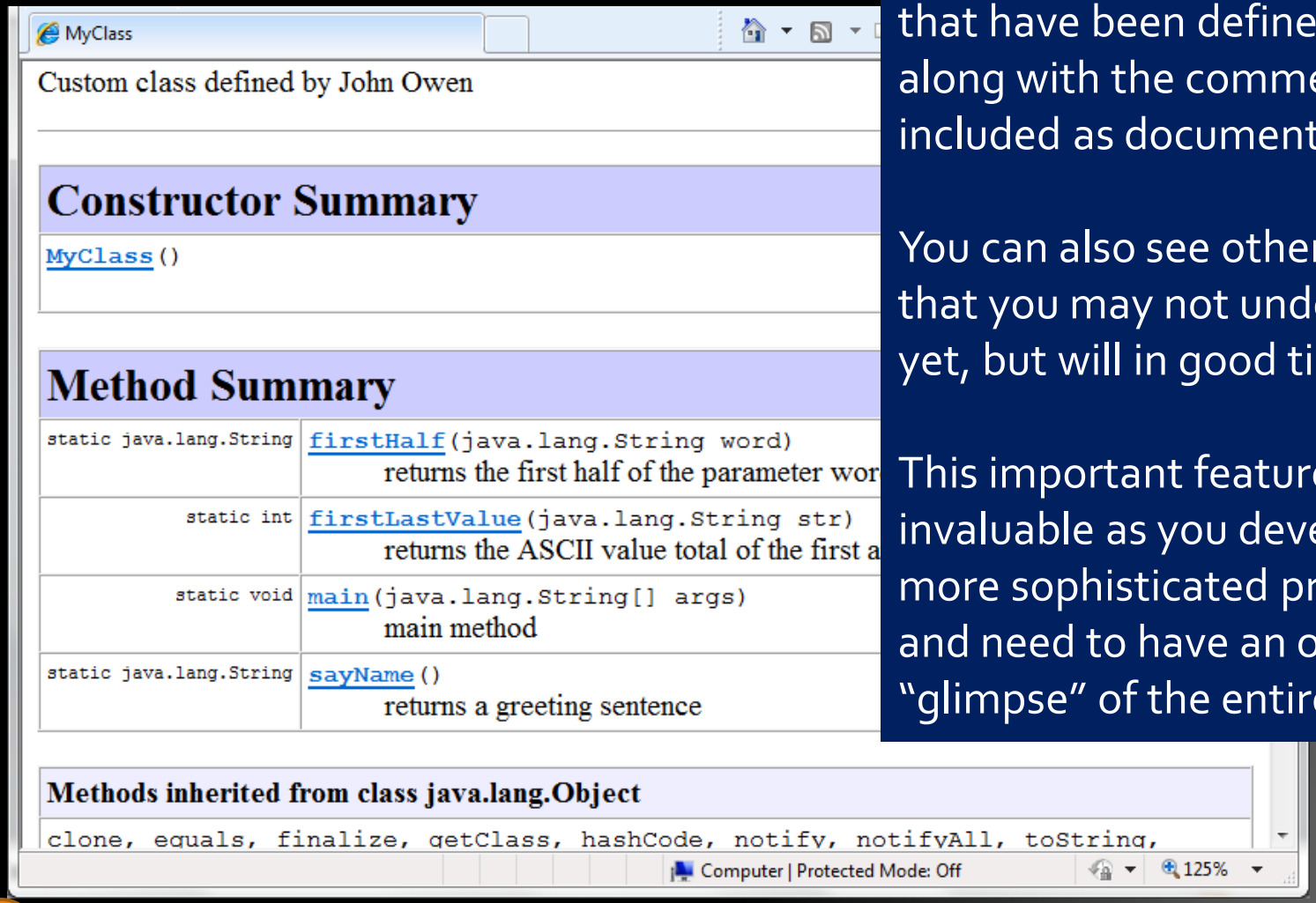
JavaDocs for JCreator

You now have a complete help file of all the work you have done so far!!!

Cool, huh!!!???



JavaDocs for JCreator



You can see the four methods that have been defined so far, along with the comments included as documentation.

You can also see other "stuff" that you may not understand yet, but will in good time.

This important feature is invaluable as you develop more sophisticated projects and need to have an organized "glimpse" of the entire project.

Lesson Summary

- In this lesson, you learned how to create your own class and define your own static return methods.
- You also learned how to create your own help file using the JavaDoc utility within the JCreator IDE.

Labs

- Now you will add more utility methods to your class, some String related, and some Math related.
- When possible, use methods already defined to help write the new method, either from the Math class, String class, or one previously defined from your own class.

Method 1 - *upperLower(String word)*

WAM (write a method) called ***upperLower*** that returns a received String with the first half uppercased and the last half lowercased.

MyClass.java | myClass1.in

```
1 rhinoceros
2 elephants
3 dogs
4 antidisestablishmentarianism
5
```

```
rhinoceros ==> RHINOceros
elephants ==> ELEPhants
dogs ==> DOgs
antidisestablishmentarianism ==> ANTIDISESTABLIshmentarianism
Press any key to continue
```

```
59      *and the last half lowercased
60      */
61     public static String upperLower(String word)
62     {
63         int half = word.length()/2;
64         String upper = word.substring(0,half).toUpperCase();
65         String lower = word.substring(half).toLowerCase();
66         return upper+lower;
67     }
68     /**
69     public static void main (String [] args) throws IOException
70     {
71         Scanner f = new Scanner(new File("myClass1.in"));
72         String str;
73         while(f.hasNext())
74         {
75             str=f.next();
76             System.out.printf("%s ==> %s\n",
77                 str,upperLower(str));
78         }
79     }
80 }
```

Method 2 – *diffWords(String one,String two)*

WAM called ***diffWords*** that receives two Strings and returns an integer that represents the positive difference between the lengths of two Strings.

MyClass.java | myClass2.in

```
1 rhinoceros
2 elephants
3 dogs
4 antidisestablishmentarianism
5
```

MyClass.java | myClass2.in

```
59      /**returns an integer that represents the positive difference
60       *between the lengths of two words
61       */
62      public static int diffWords(String one, String two)
63      {
64          int diff;
65          if(one.length()>two.length())
66              diff=one.length()-two.length();
67          else
68              diff=two.length()-one.length();
69          return diff;
70      }
71      public static void main (String [] args) throws IOException
72      {
73          Scanner f = new Scanner(new File("myClass2.in"));
74          String s1,s2;
75          while(f.hasNext())
76          {
77              s1=f.next();
78              s2=f.next();
79              System.out.printf("%s %s Length difference = %s\n",
80                  s1,s2,diffWords(s1,s2));
81          }
82      }
83  }
```

```
rhinoceros elephants Length difference = 1
dogs antidisestablishmentarianism Length difference = 24
Press any key to continue...
```

Method 3 – *sameLength(String one,String two)*

WAM called ***sameLength*** that receives two Strings and returns a boolean value to indicate if two words are the same length.

MyClass.java | myClass3.in

```
/**returns true or false whether or not two words
 *are the same length
 */
public static boolean sameLength(String one, String two)
{
}

public static void main (String [] args) throws IOException
{
    Scanner f = new Scanner(new File("myClass3.in"));
    String s1,s2;
    while(f.hasNext())
    {
        s1=f.next();
        s2=f.next();
        System.out.printf("%s %s Same length = %s\n",
            s1,s2,sameLength(s1,s2));
    }
}
```

MyClass.java | myClass3.in

```
horses
pigs
cows
dogs
```

```
horses pigs Same length = false
cows dogs Same length = true
Press any key to continue...
```

Method 4 – *areaCircle(double radius)*

WAM called ***areaCircle*** that receives a radius as a double value and returns the area of a circle with the given radius.

```
Radius = 3.1 : Area = 30.97
Radius = 10.0 : Area = 314.16
Press any key to continue...
```

MyClass.java | myClass4.in

```
3.14
10
```

MyClass.java | myClass4.in

```
60      /**returns the area of a circle given the radius
61       */
62      public static double areaCircle(double radius)
63      {
64
65
66
67      }
68
69      public static void main (String [] args) throws IOException
70      {
71          Scanner f = new Scanner(new File("myClass4.in"));
72          double rad;
73          while(f.hasNext())
74          {
75              rad = f.nextDouble();
76              System.out.printf("Radius = %.1f : Circle Area = %.2f\n",
77                  rad,areaCircle(rad));
78          }
79      }
80  }
```

Method 5 – *surfAreaSphere(double radius)*

WAM called ***surfAreaSphere*** that receives a radius and returns the surface area of a sphere with the given radius. Use the ***areaCircle*** method you wrote previously to help with this one.

MyClass.java | myClass5.in

```
/**returns the surface area of a sphere given the radius
 */
public static double surfAreaSphere(double radius)
{
```

myClass5.in

```
3.14
10
```

```
Radius = 3.1 : Surface Area = 123.90
Radius = 10.0 : Surface Area = 1256.64
Press any key to continue...
```

```
public static void main (String [] args) throws IOException
{
    Scanner f = new Scanner(new File("myClass5.in"));
    double rad;
    while(f.hasNext())
    {
        rad = f.nextDouble();
        System.out.printf("Radius = %.1f : Surface Area = %.2f\n",
            rad,surfAreaSphere(rad));
    }
}
```

Method 6 –
delta(int num1, int num2)

WAM called ***delta*** that returns the difference between two received integers. *Delta "Δ" is the Greek letter mathematicians use to refer to* ***difference****.*

MyClass.java | myClass6.in

```
1  3 5 6 10
2  -6 3 -6 7
3  4 -5 1 -5
```

```
delta(3,5)==> -2
delta(6,10)==> -4
delta(-6,3)==> -9
delta(-6,7)==> -13
delta(4,-5)==> 9
delta(1,-5)==> 6
Press any key to contin
```

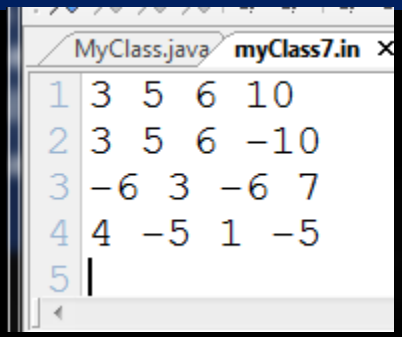
MyClass.java | myClass6.in

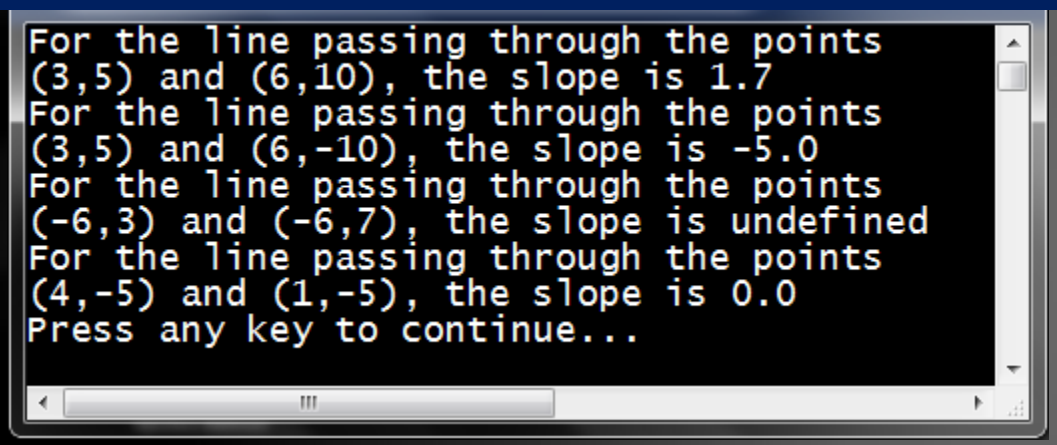
```
74      /**returns the difference between two integers
75       */
76      public static int delta(int num1, int num2)
77      {
78          return num1-num2;
79      }
80
81
82      public static void main (String [] args) throws IOException
83      {
84          Scanner f = new Scanner(new File("myClass6.in"));
85          int n1,n2;
86          while(f.hasNext())
87          {
88              n1=f.nextInt();
89              n2=f.nextInt();
90              System.out.printf("delta(%d,%d)==> %d\n",
91                  n1,n2,delta(n1,n2));
92          }
93      }
94  }
```

Method 7 –
slope(int x1,int y1, int x2, int y2)

WAM called ***slope*** that returns the slope of a line as a String, given four integers representing two points. If the slope is undefined, return the word "undefined", otherwise return a decimal value formatted to 1 decimal place, **but as a String.**

Hint: You will need to use an if statement and the String.format command (see Lesson 1C).





Method 7 –
slope(int x1,int y1, int x2, int y2)

MyClass.java | myClass7.in

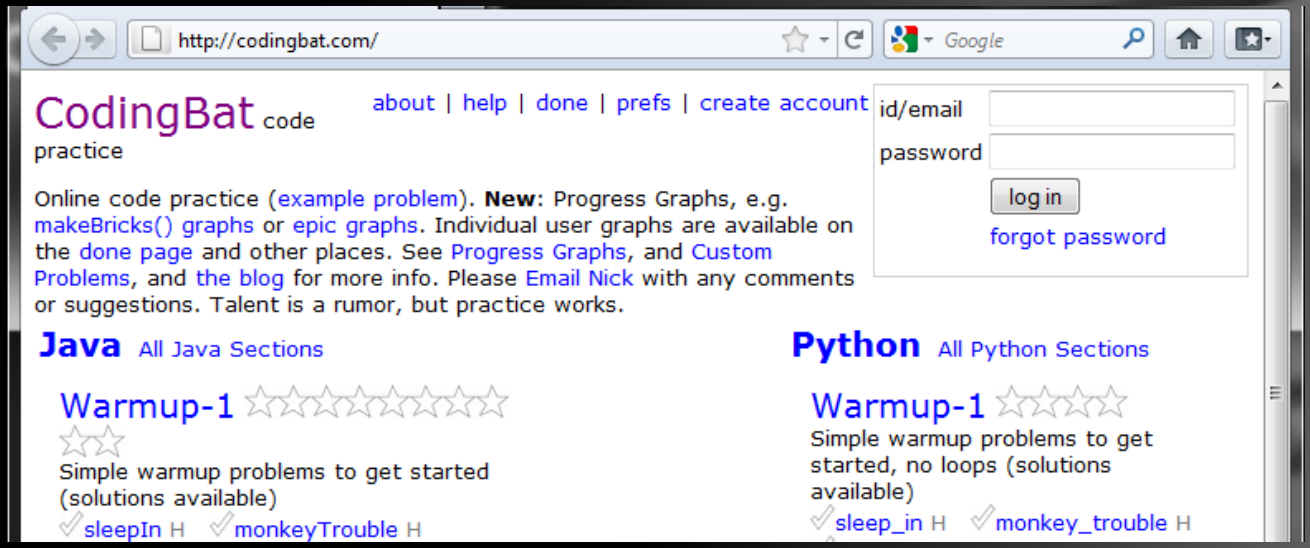
```
72      /**returns as a String the slope of a line given four integers
73       *representing two points in a coordinate plane,
74       *formatted to one decimal place.
75       *Return "undefined" if appropriate
76       */
77      public static String slope(int x1,int y1, int x2, int y2)
78      {
79
80
81
82      }
83      /
84
85      public static void main (String [] args) throws IOException
86      {
87          Scanner f = new Scanner(new File("myClass7.in"));
88          int x1,y1,x2,y2;
89          while(f.hasNext())
90          {
91              x1=f.nextInt();
92              y1=f.nextInt();
93              x2=f.nextInt();
94              y2=f.nextInt();
95              System.out.printf("For the line passing through the points\n"
96                  +"(%d,%d) and (%d,%d), the slope is %s\n",
97                  x1,y1,x2,y2, slope(x1,y1,x2,y2));
98          }
99      }
100 }
```

JavaDoc Review

- Once you have completed these first seven methods, run the JavaDoc tool and see the documentation for the work you did.
- Consider revising the documentation if you feel it needs clarification or adjustment.

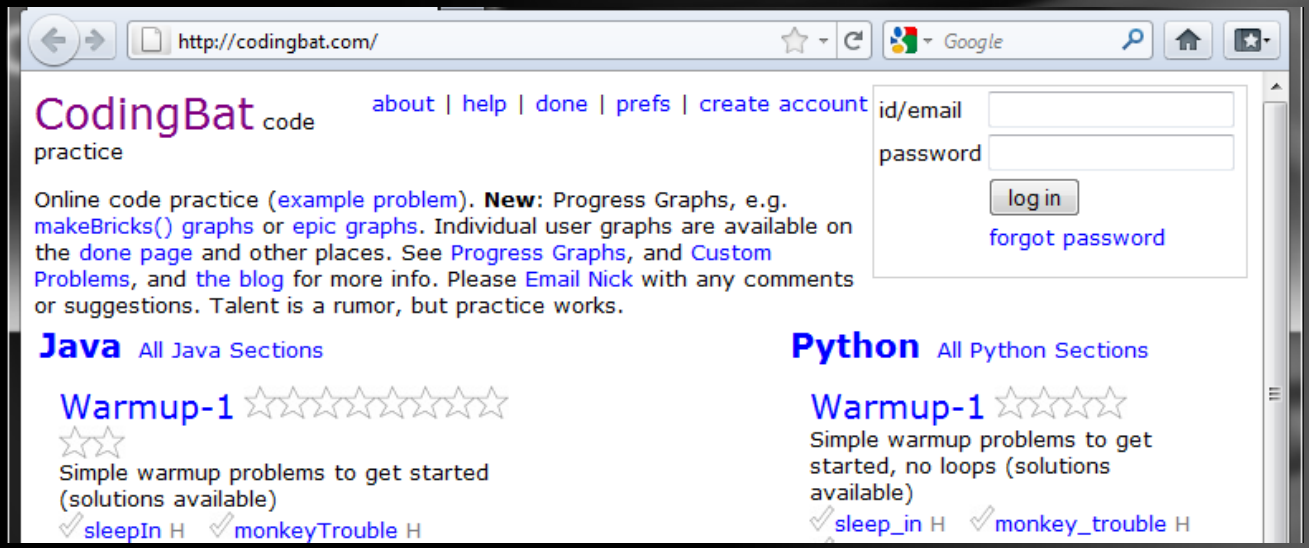
CodingBat Labs

- These next three methods will be submitted using a Java practice web site called codingbat.com.



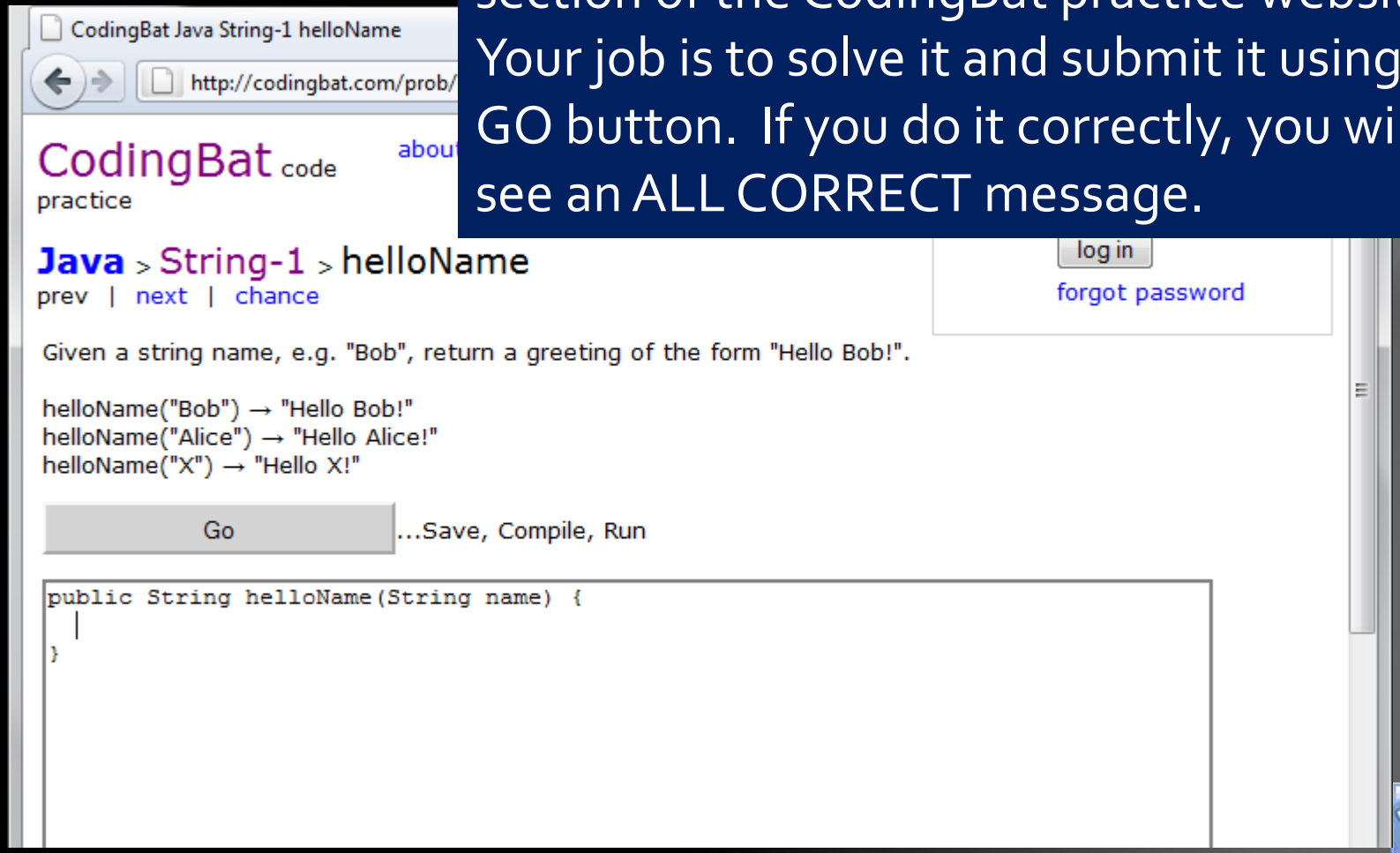
CodingBat Labs

- To use this website you need to register (free) with an email address.



Method 8 - *helloName*

This is the first example in the `String-1` section of the CodingBat practice website. Your job is to solve it and submit it using the GO button. If you do it correctly, you will see an ALL CORRECT message.

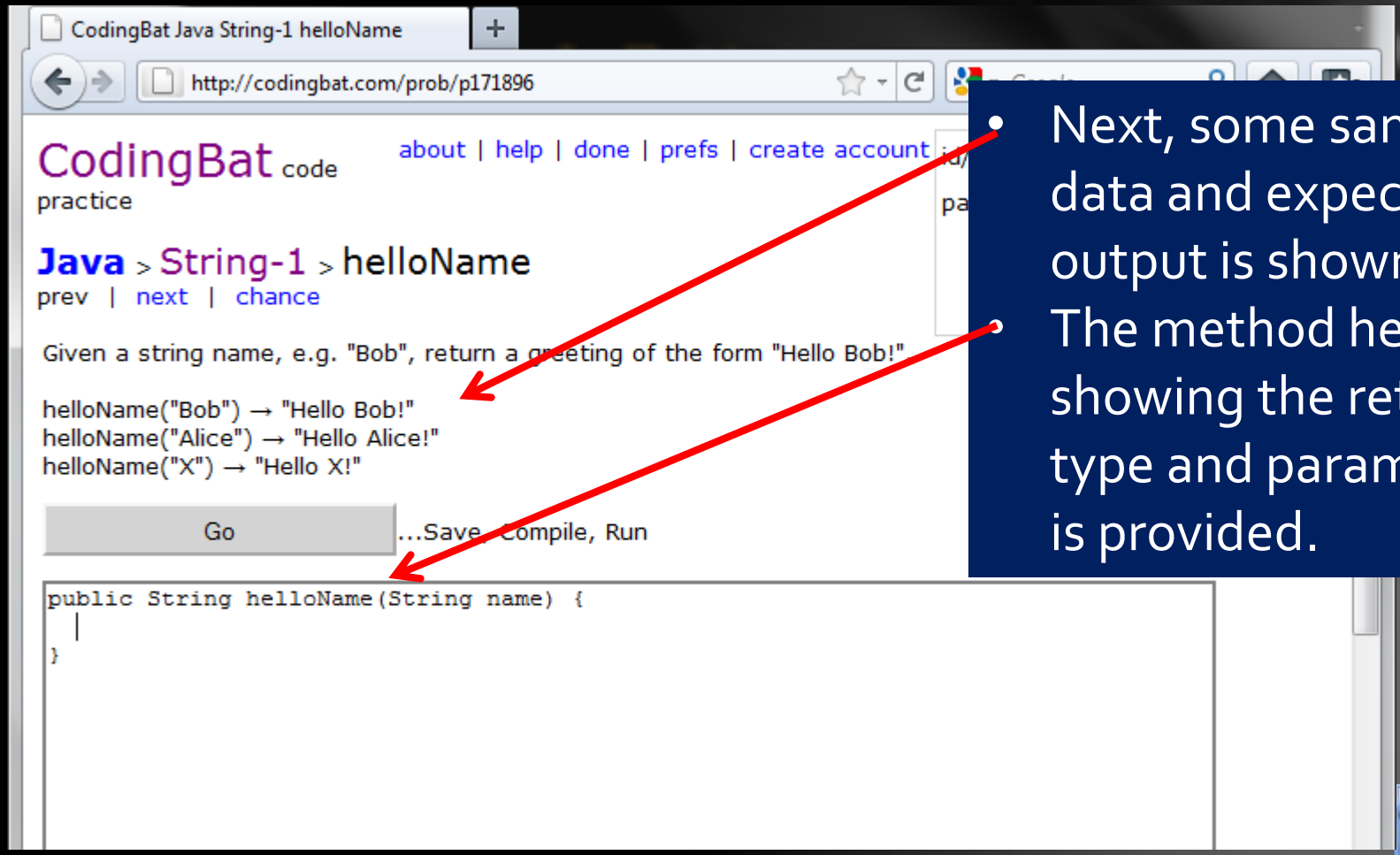


Method 8 - *helloName*

This method is very easy, and represents the basic process CodingBat uses. Notice several important features.

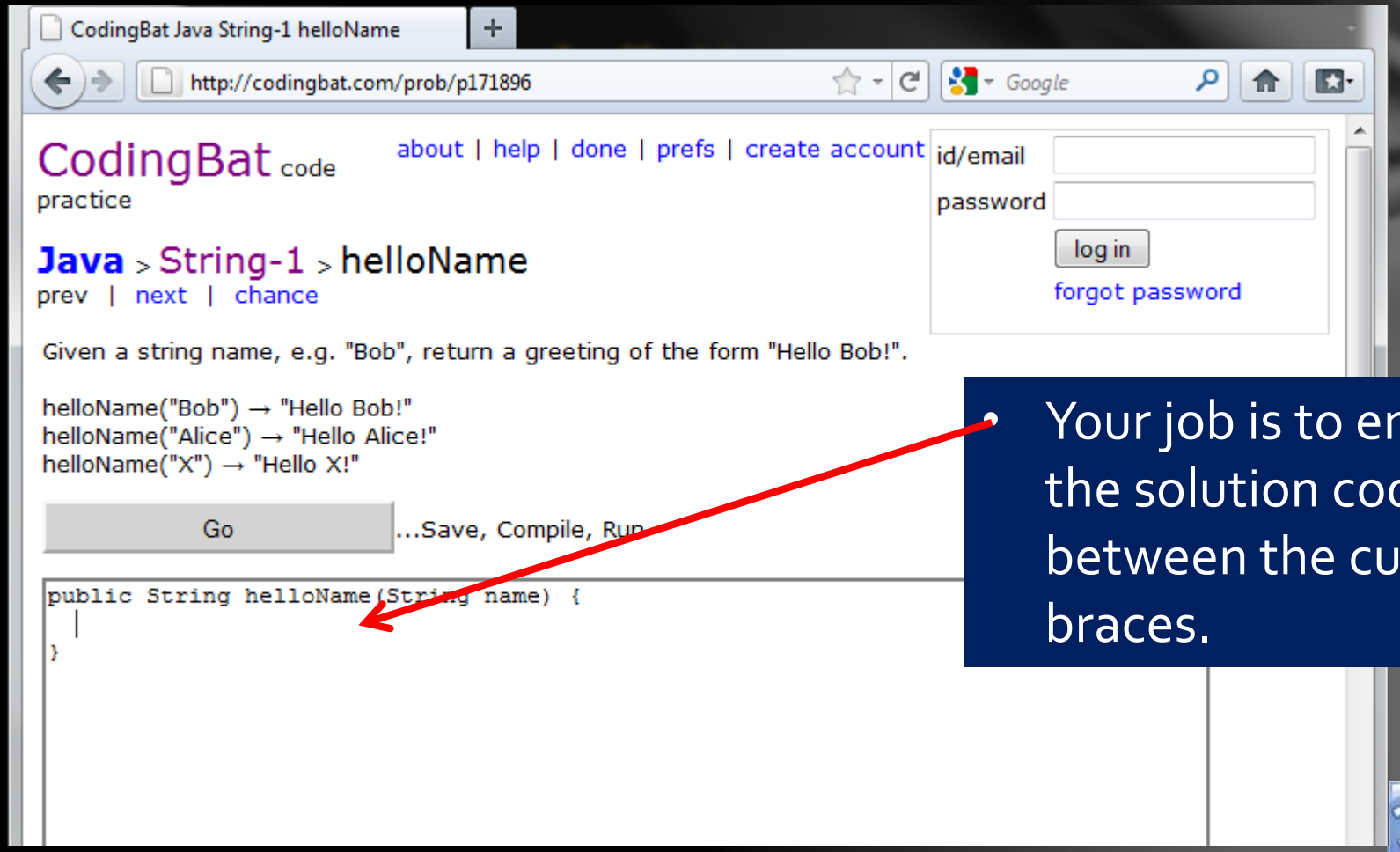
- First, a description of the method and its expected outcome is given.

Method 8 - *helloName*



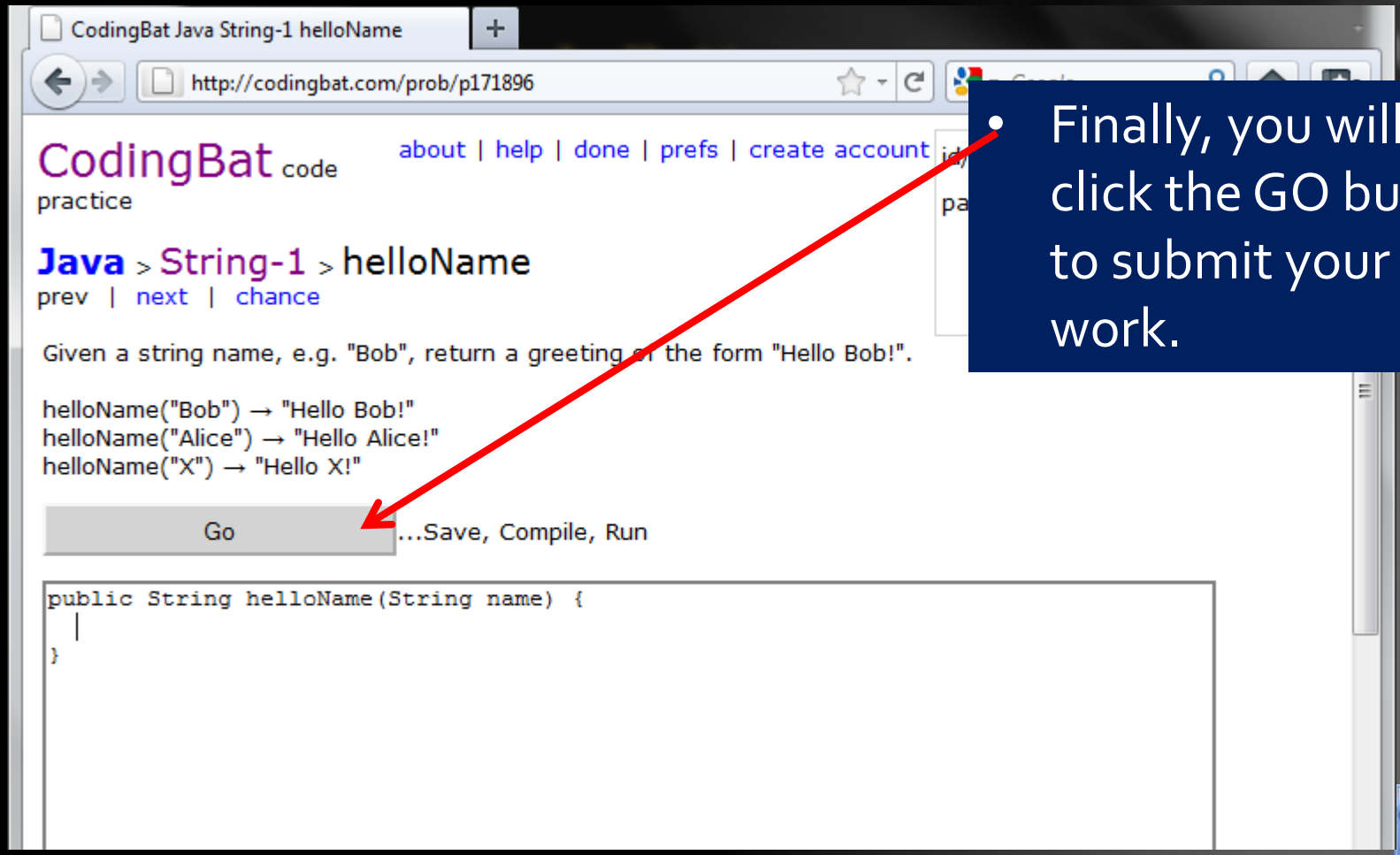
- Next, some sample data and expected output is shown.
- The method header showing the return type and parameter is provided.

Method 8 - *helloName*



- Your job is to enter the solution code between the curly braces.

Method 8 - *helloName*

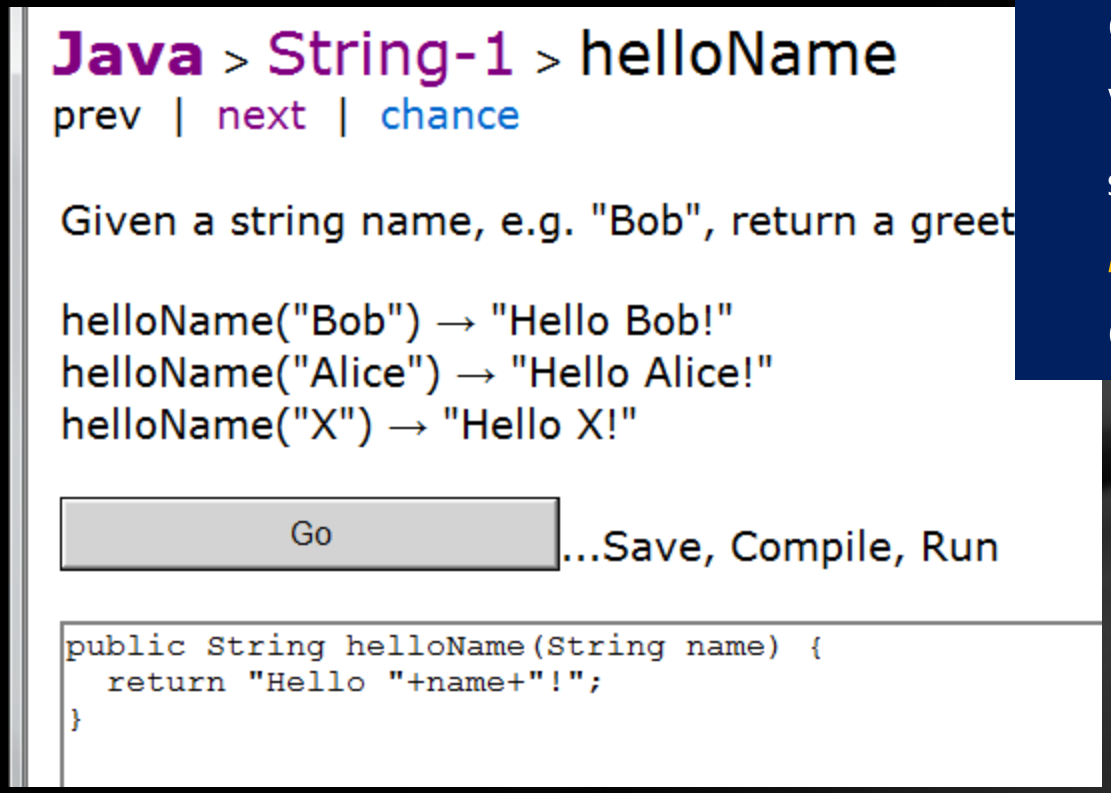


- Finally, you will click the GO button to submit your work.

Method 8 - *helloName*

- Try it yourself first, and if you need to, take a look at the solution on the next slide.

Method 8 - *helloName*



- The solution is a simple concatenation of the word "Hello " (note the space), the parameter ***name***, and the exclamation mark.

Method 8 - *helloName*

- When you click the GO button, you will receive a report like the one shown below, hopefully with the words, "All Correct".

Java > String-1 > helloName

prev | next | chance

Given a string name, e.g. "Bob", return a greeting

helloName("Bob") → "Hello Bob!"
helloName("Alice") → "Hello Alice!"
helloName("X") → "Hello X!"

Go ...Save, Compile,

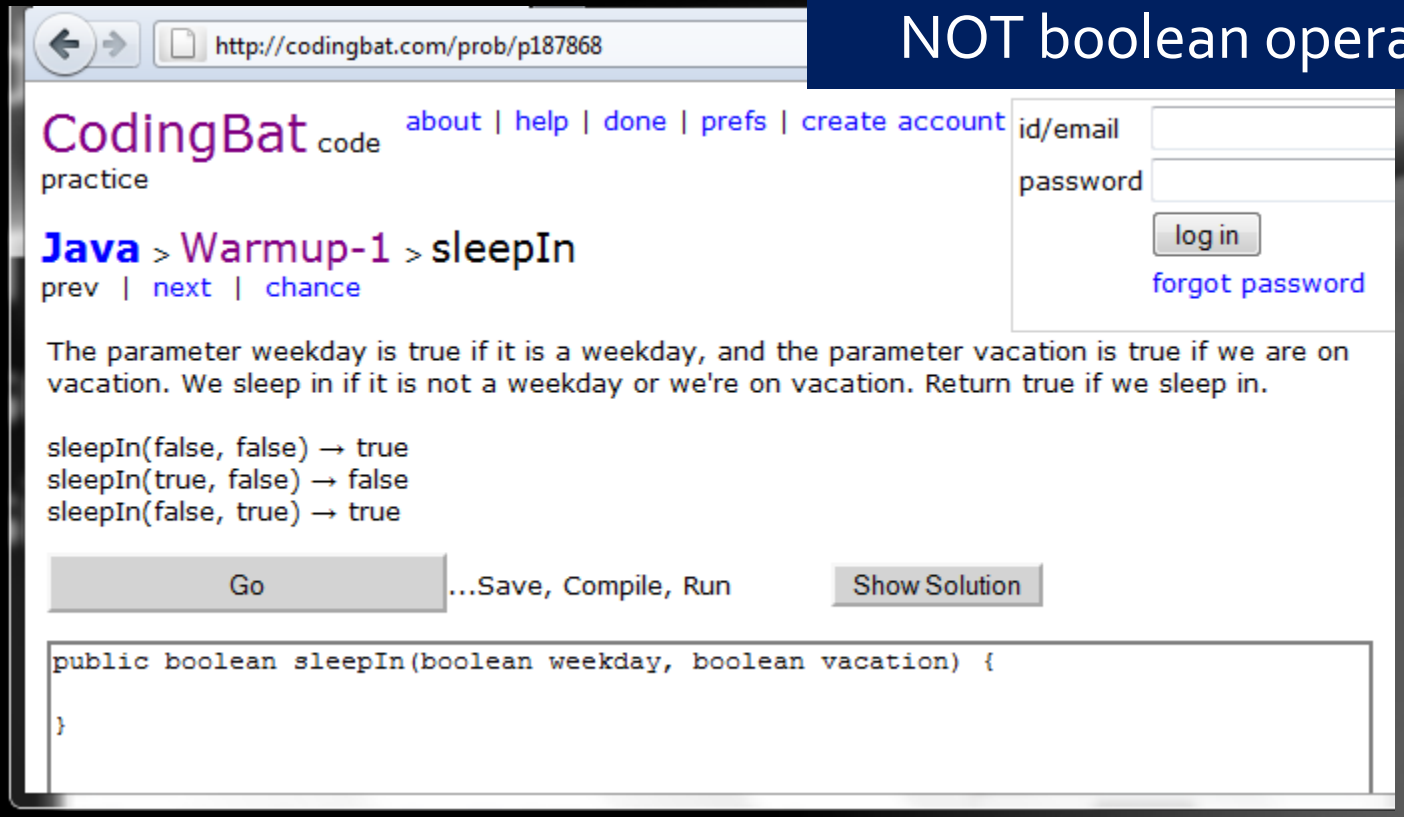
```
public String helloName(String name) {
  return "Hello "+name+"!";
}
```

Expected	This Run	
helloName("Bob") → "Hello Bob!"	"Hello Bob!"	OK
helloName("Alice") → "Hello Alice!"	"Hello Alice!"	OK
helloName("X") → "Hello X!"	"Hello X!"	OK
helloName("Hello") → "Hello Hello!"	"Hello Hello!"	OK

All Correct

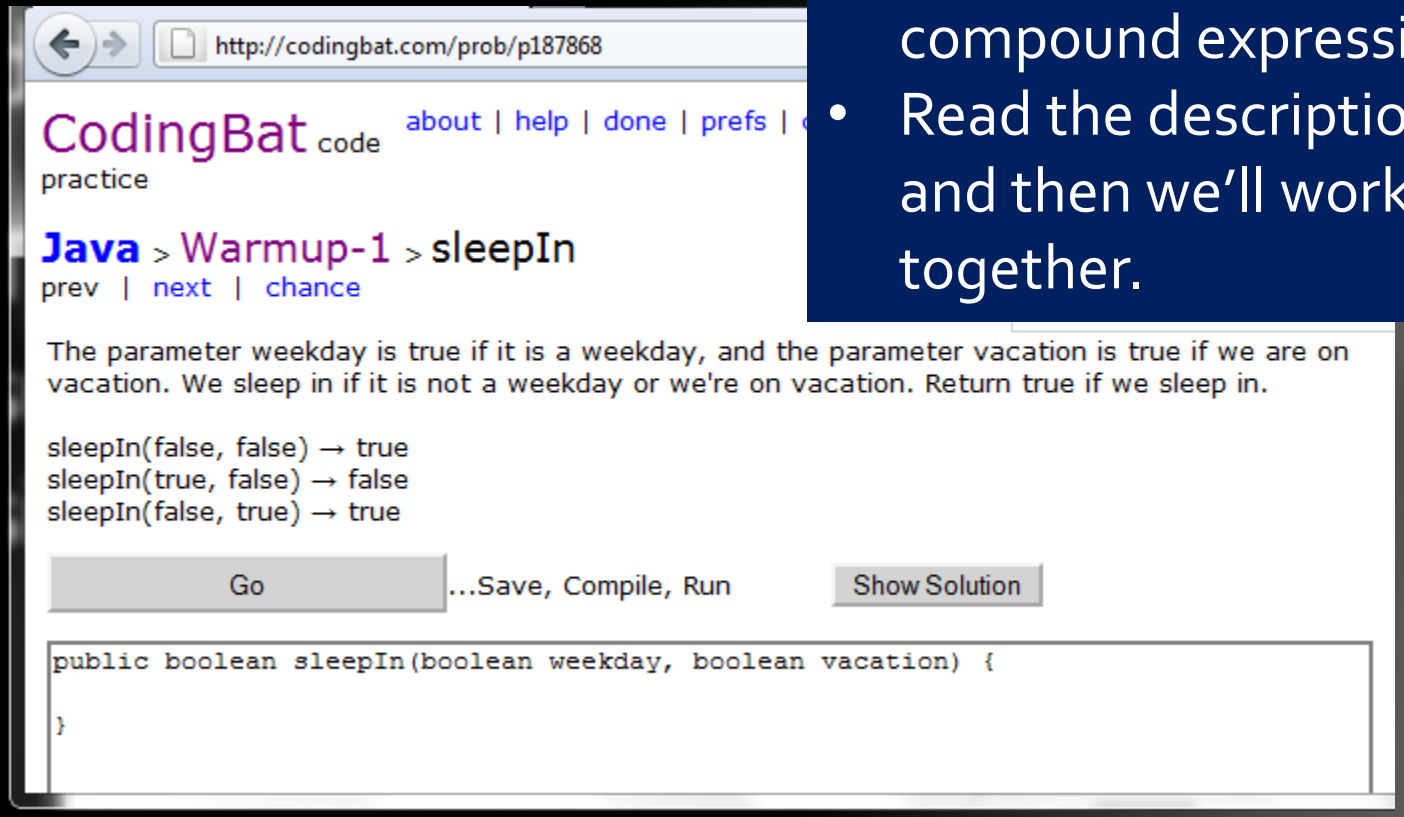
Method 9 - *sleepIn*

- This next method is the first one in the **`Warmup-1`** section of the CodingBat website.
- We will work through several ways to solve this problem and introduce the AND, OR, and NOT boolean operators.



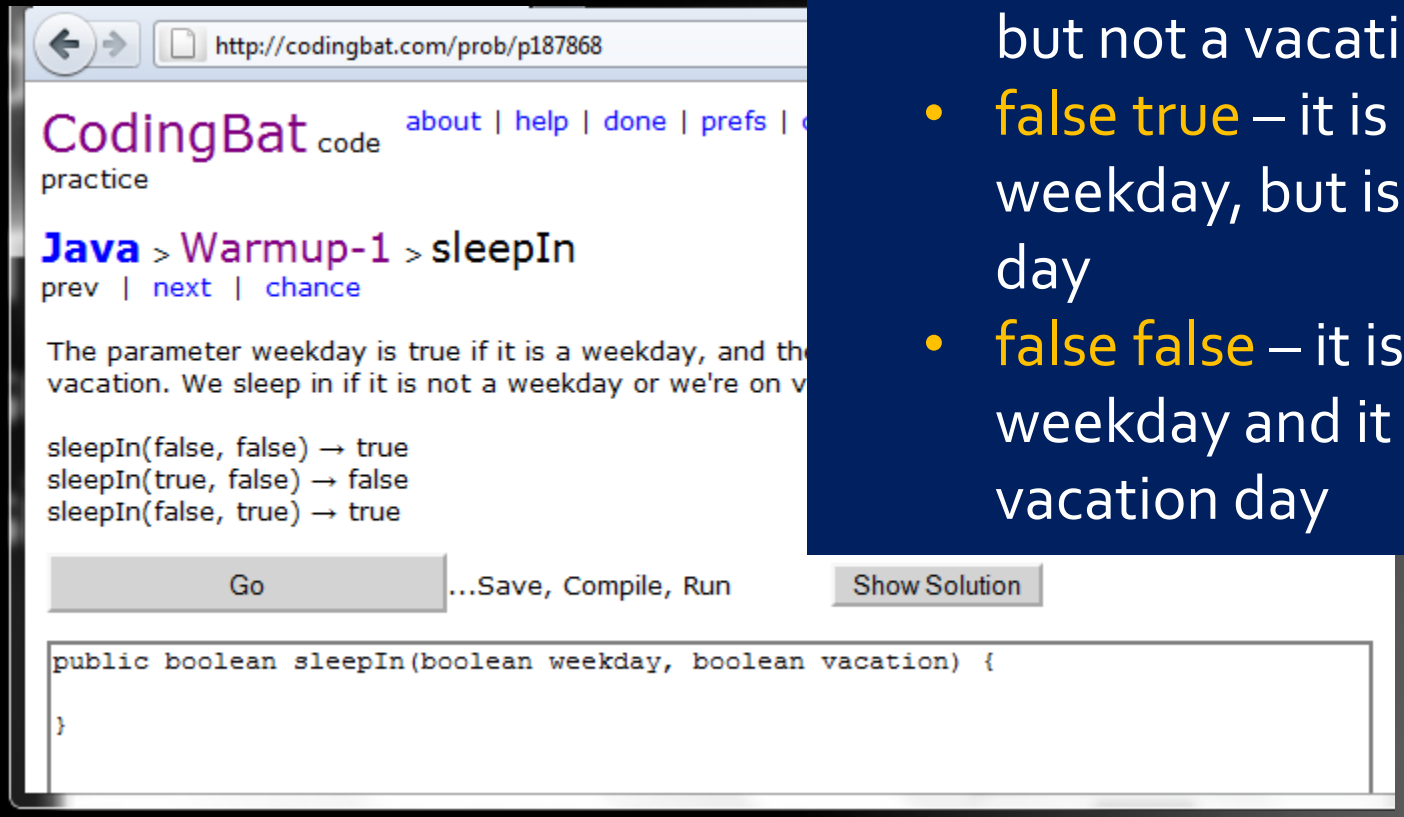
Method 9 - *sleepIn*

- In this method, you will return a boolean value based on two other boolean values, and you will use a new feature called AND to connect two boolean expressions into one compound expression.
- Read the description carefully, and then we'll work through it together.



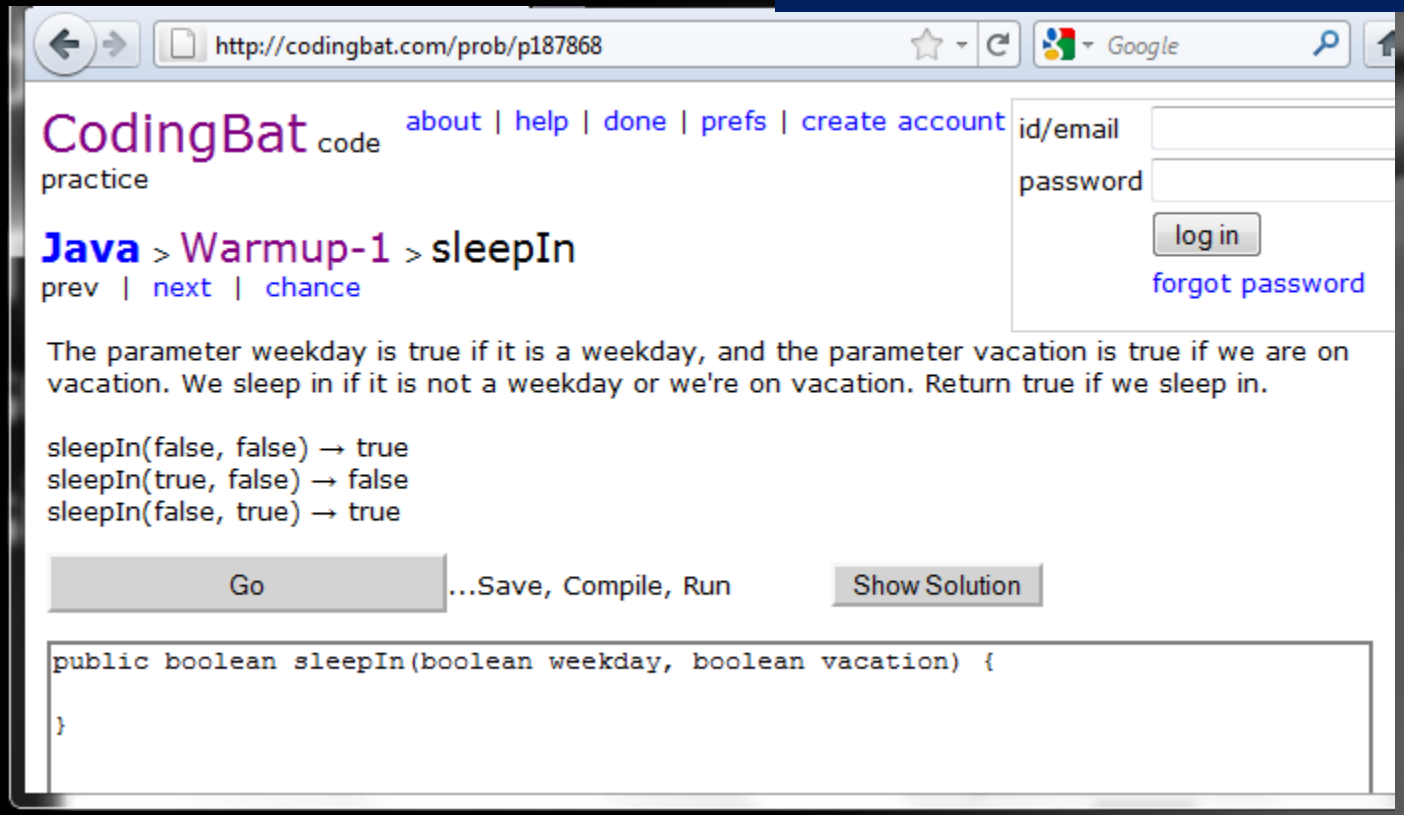
Method 9 - *sleepIn*

- There are only four possible combinations of the two boolean parameters:
  - true true – it is a weekday and it is a vacation day
  - true false – it is a weekday, but not a vacation day
  - false true – it is not a weekday, but is a vacation day
  - false false – it is not a weekday and it is not a vacation day



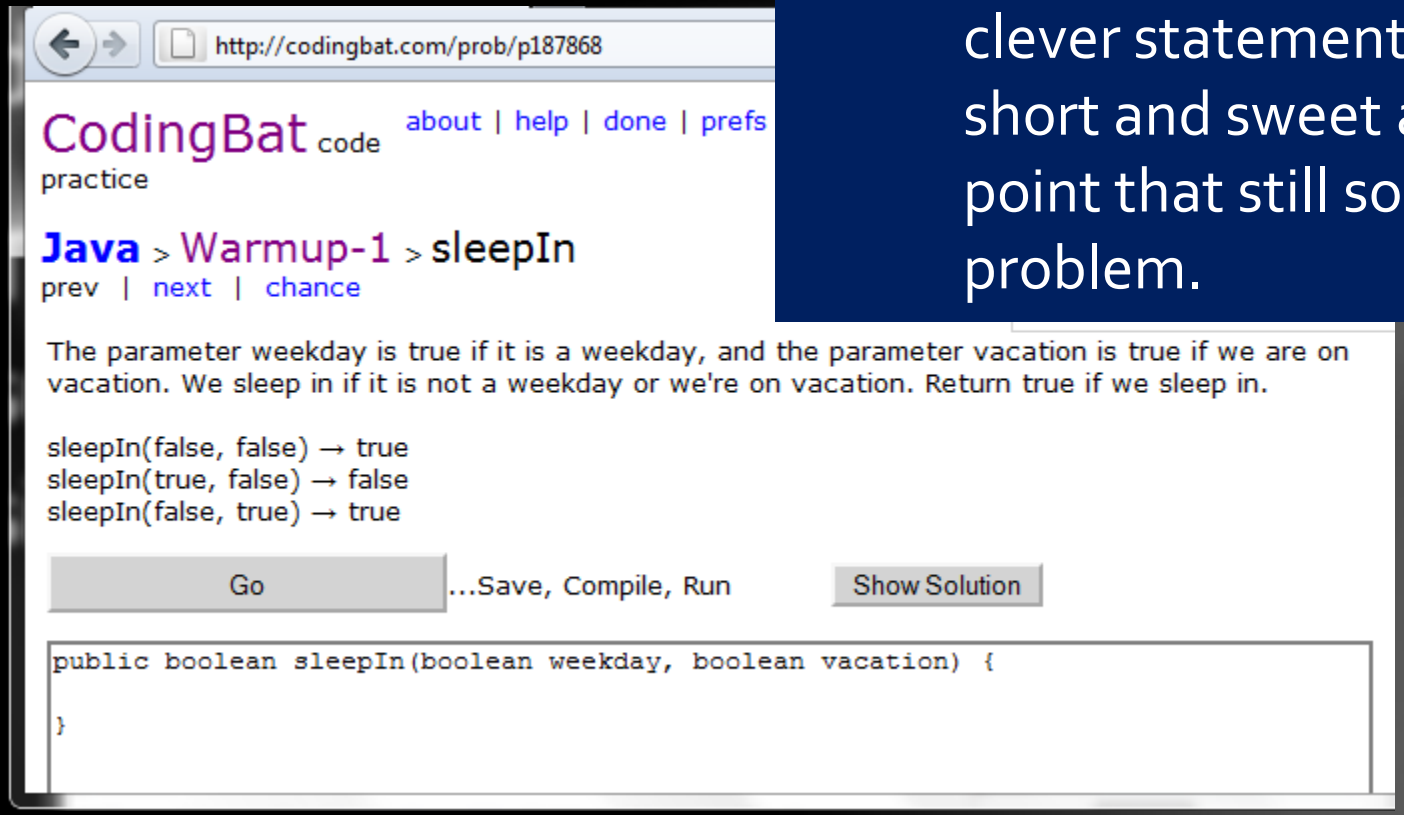
Method 9 - *sleepIn*

- Your job is to design a return statement, or combination of several return statements, that will correctly return true or false, indicating whether or not you can sleep in (legally, that is).



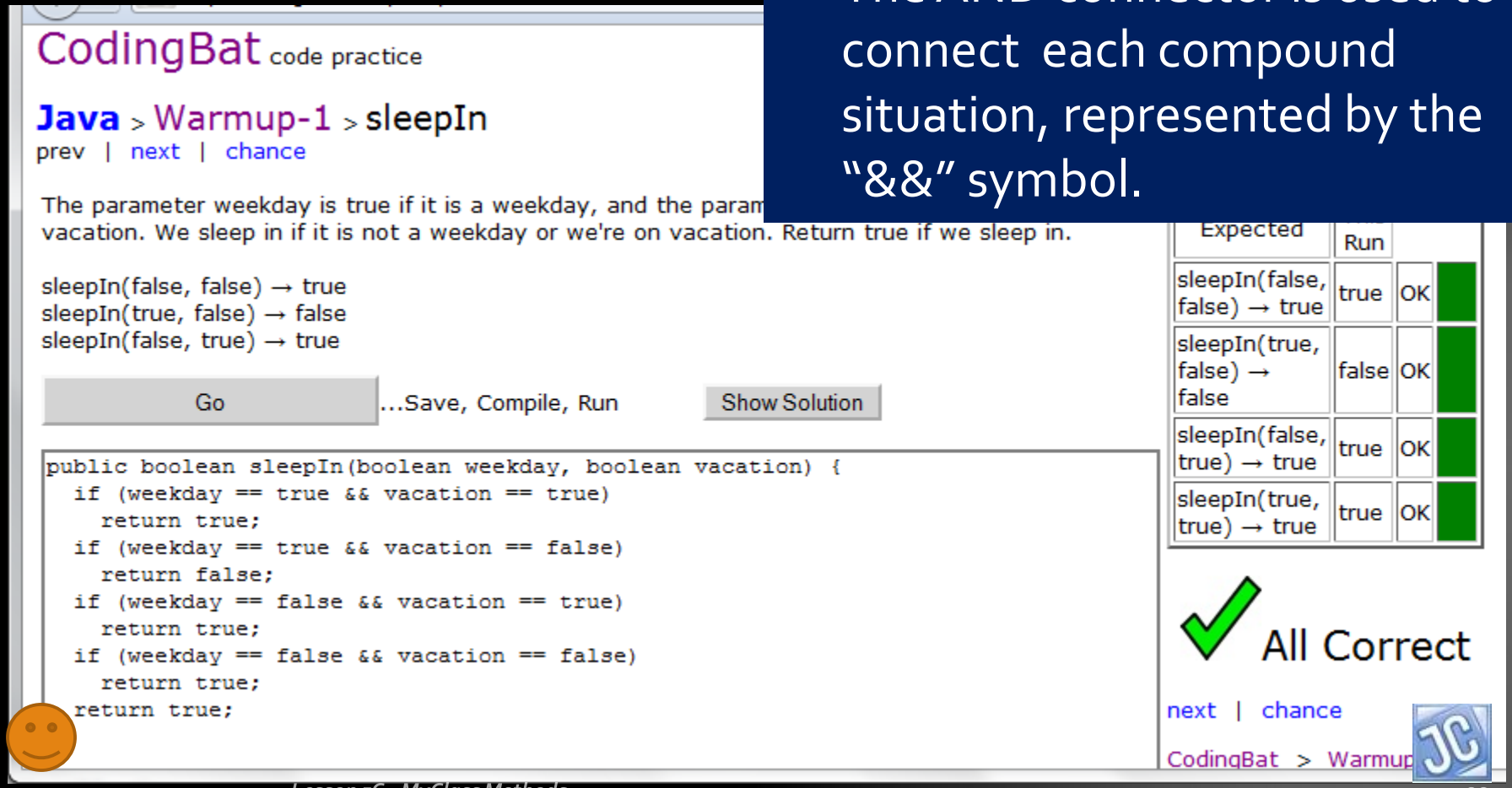
Method 9 - *sleepIn*

- There are several ways to do this:
  - The "brute force" way…set up an if statement for each possibility.
  - The "elegant" way…some clever statement that is short and sweet and to the point that still solves the problem.



Method 9 - *sleepIn*

- Below is the "brute force" method.
- The AND connector is used to connect each compound situation, represented by the "&&" symbol.



CodingBat code practice

Java > Warmup-1 > sleepIn

prev | next | chance

The parameter weekday is true if it is a weekday, and the param
vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in.

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go ...Save, Compile, Run Show Solution

```
public boolean sleepIn(boolean weekday, boolean vacation) {
  if (weekday == true && vacation == true)
    return true;
  if (weekday == true && vacation == false)
    return false;
  if (weekday == false && vacation == true)
    return true;
  if (weekday == false && vacation == false)
    return true;
  return true;
```

Expected	Run		
sleepIn(false, false) → true	true	OK	
sleepIn(true, false) → false	false	OK	
sleepIn(false, true) → true	true	OK	
sleepIn(true, true) → true	true	OK	

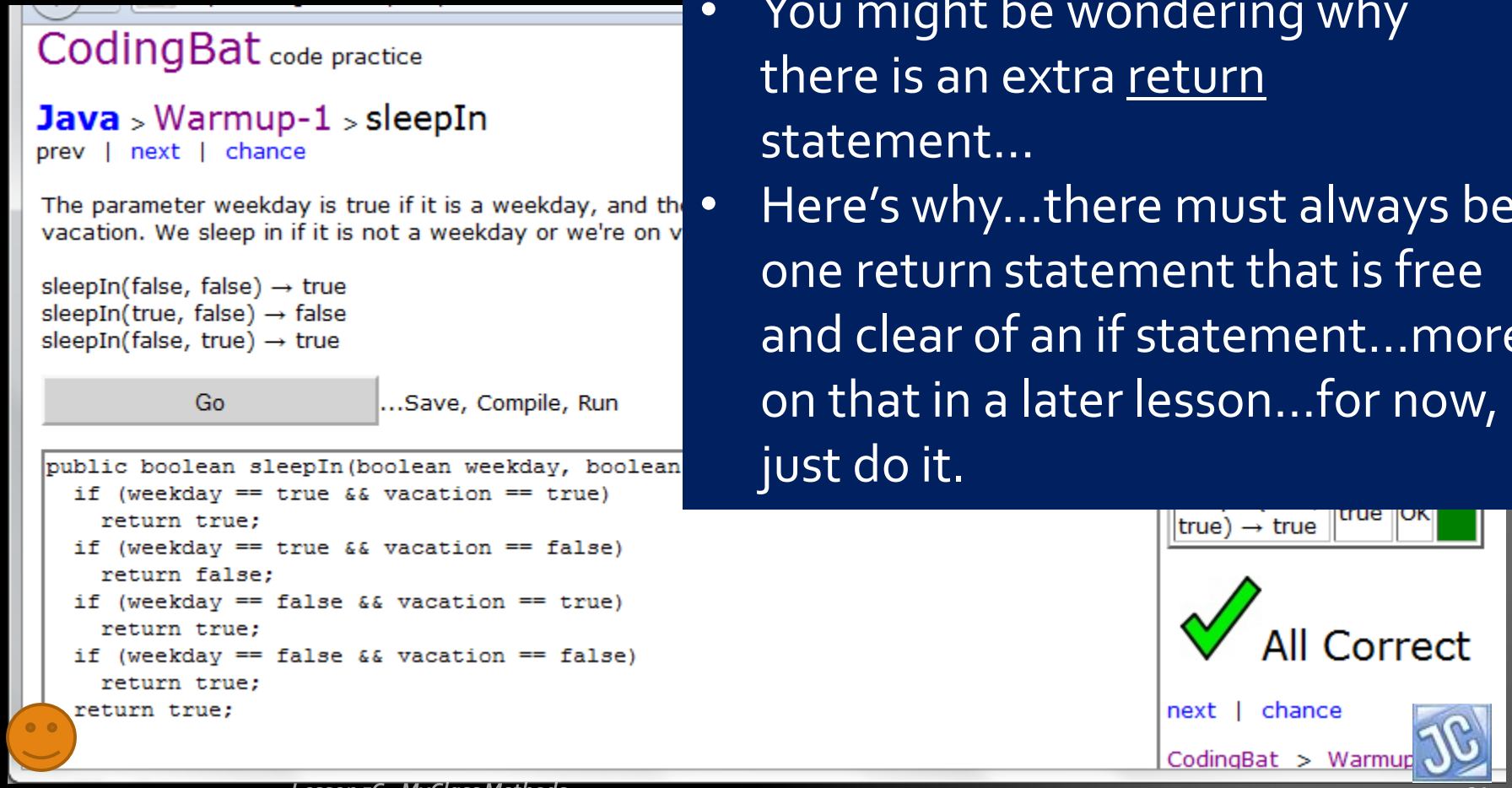
All Correct

next | chance

CodingBat > Warmup

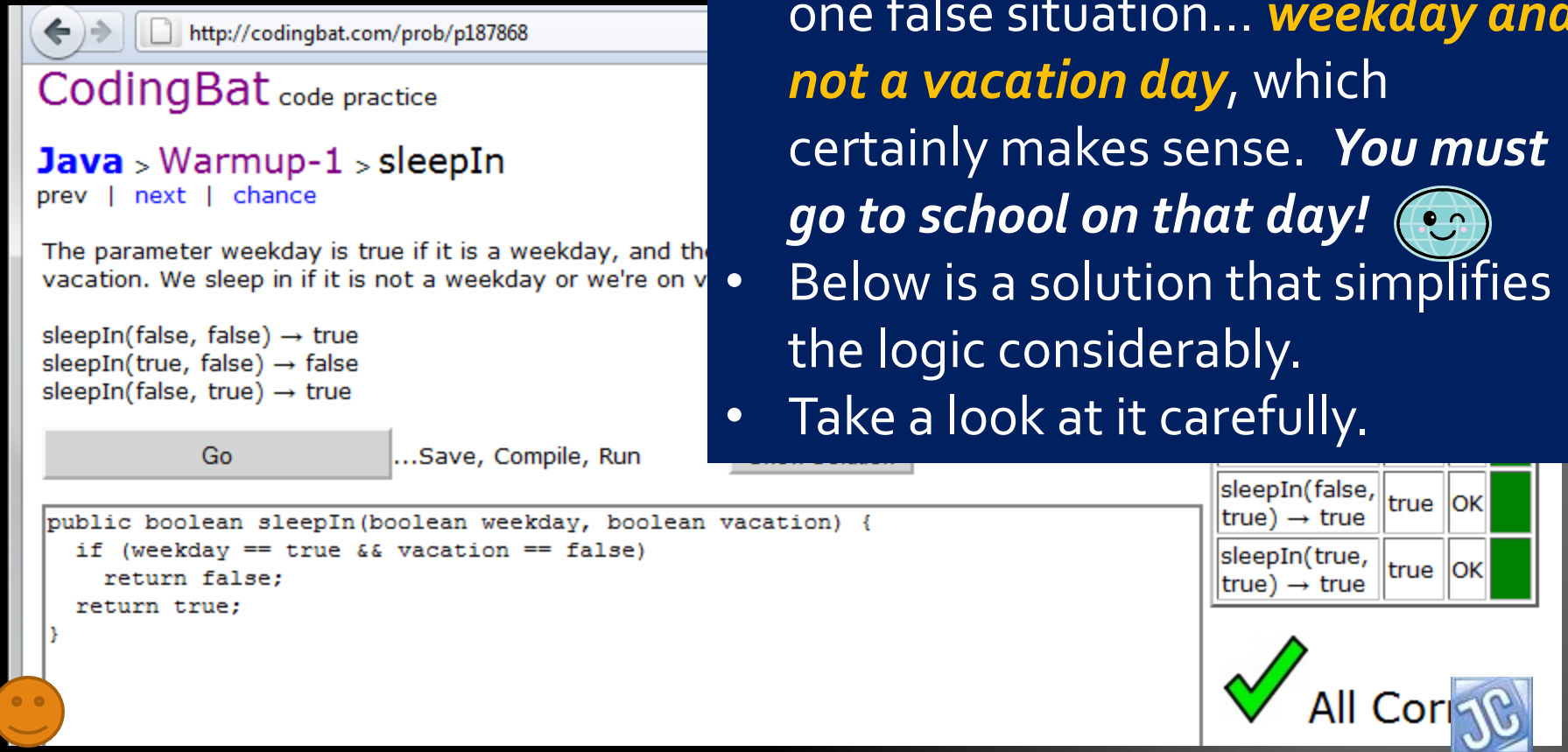
Method 9 - *sleepIn*

- Every combination is represented, and the method works.
- You might be wondering why there is an extra return statement…
- Here's why…there must always be one return statement that is free and clear of an if statement…more on that in a later lesson…for now, just do it.



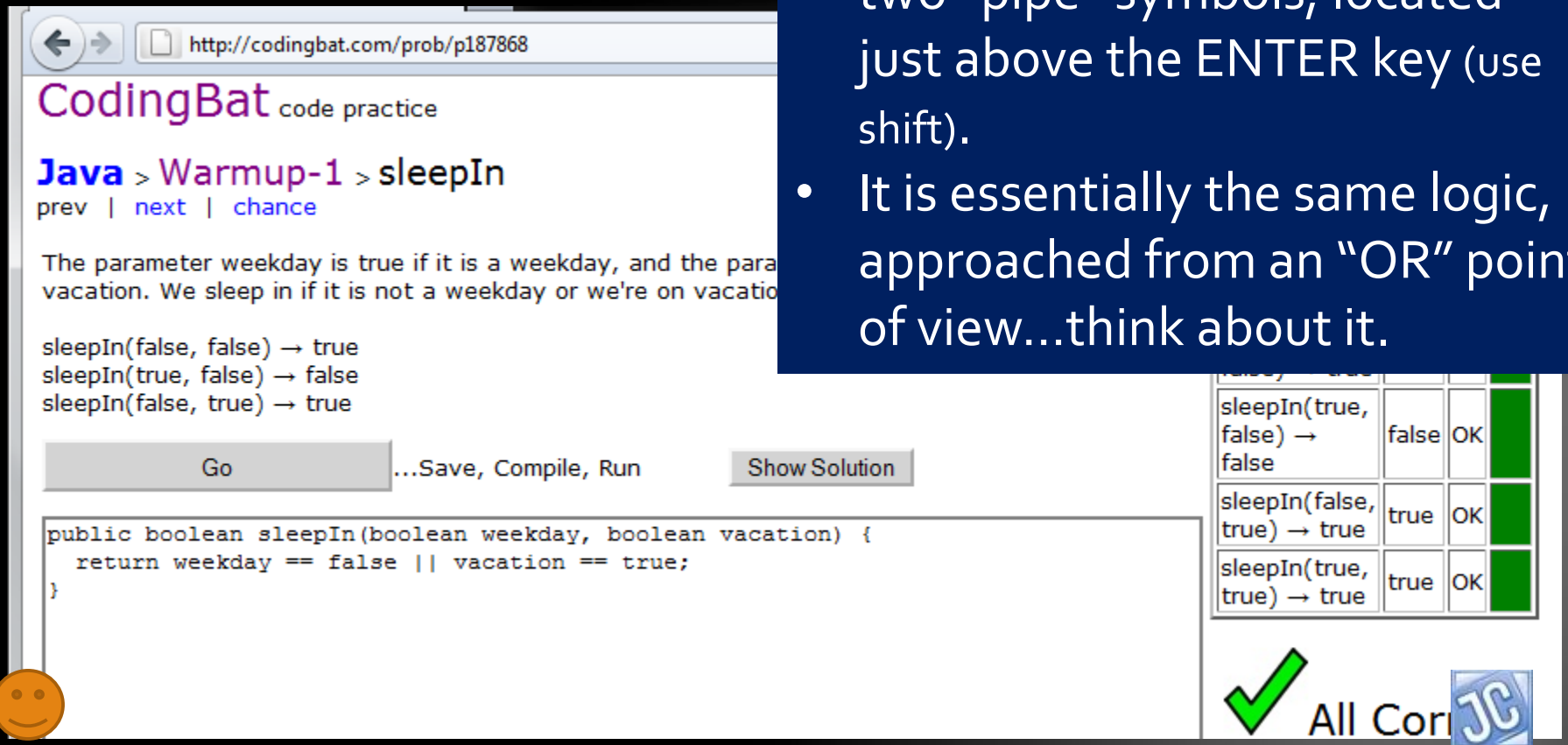
Method 9 - *sleepIn*

- There is a "more elegant" way…
- In the previous solution, you might have noticed that there was only one false situation… ***weekday and not a vacation day***, which certainly makes sense. ***You must go to school on that day!***
- Below is a solution that simplifies the logic considerably.
- Take a look at it carefully.



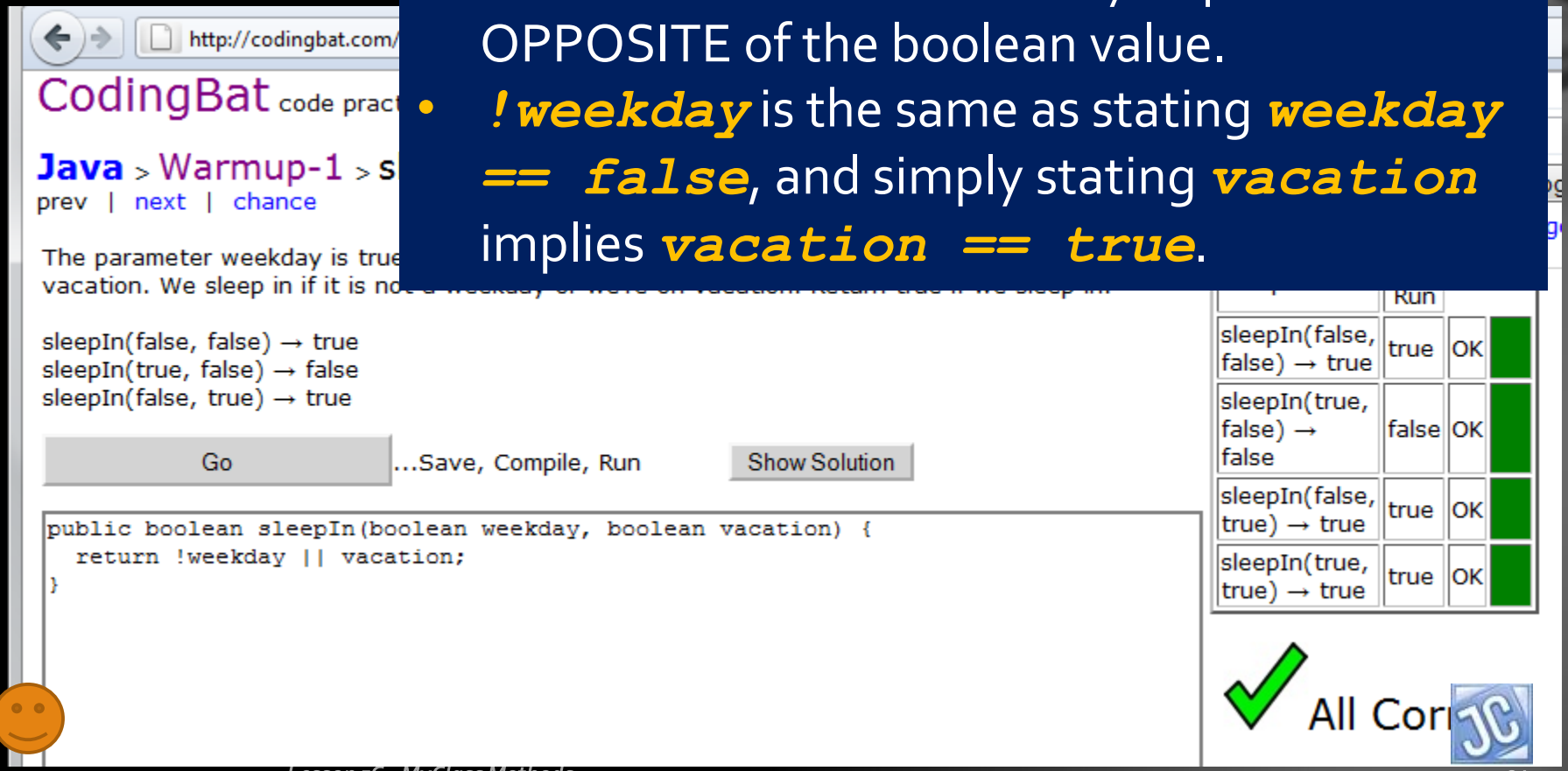
Method 9 - *sleepIn*

- Here is an even more "elegant" solution that uses the OR connector, "||", which is two "pipe" symbols, located just above the ENTER key (use shift).
- It is essentially the same logic, approached from an "OR" point of view…think about it.



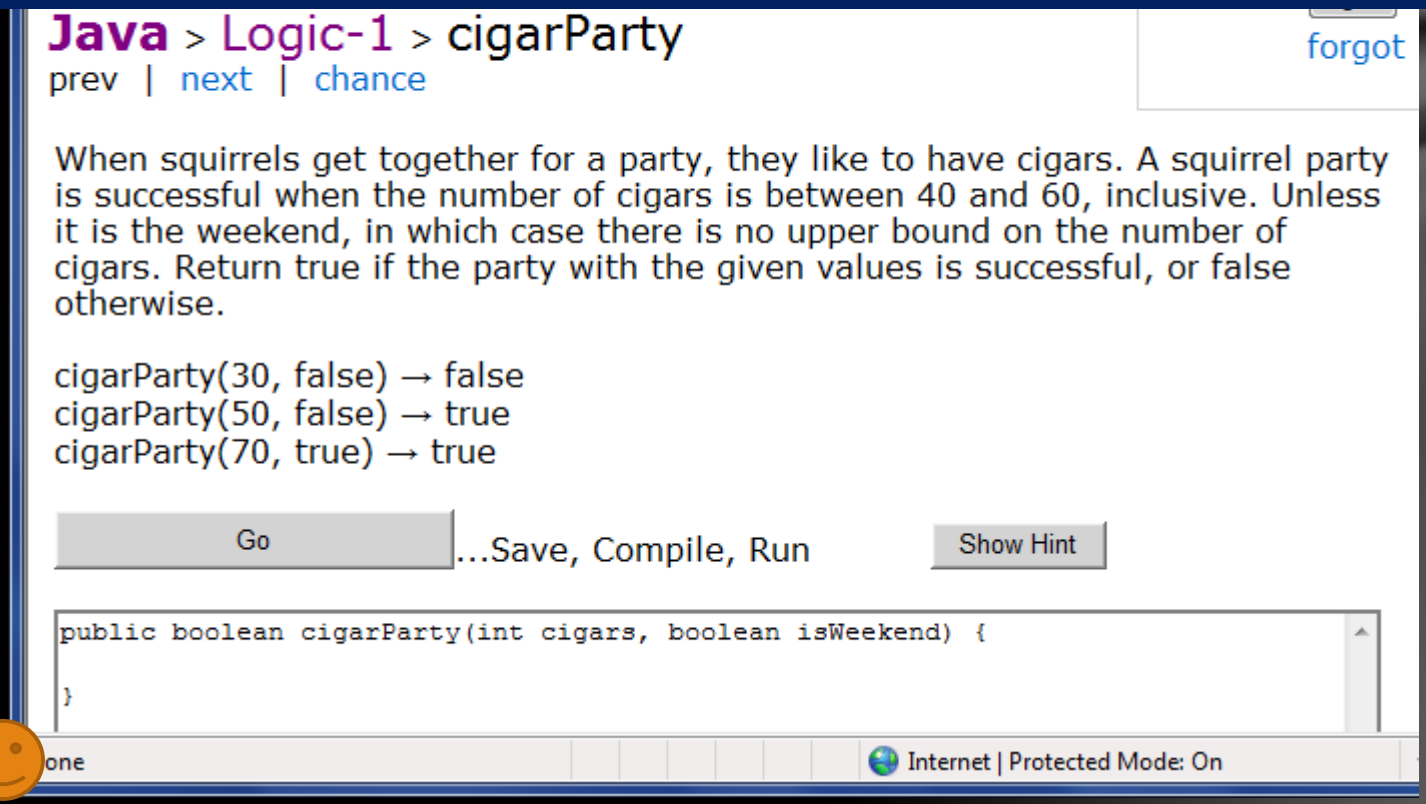
Method 9 - *sleepIn*

- Here is the "most elegant" solution that uses the OR connector and the NOT symbol, which when used in this way implies the OPPOSITE of the boolean value.
- ***!weekday*** is the same as stating ***weekday == false***, and simply stating ***vacation*** implies ***vacation == true***.



Method 10 - *cigarParty*

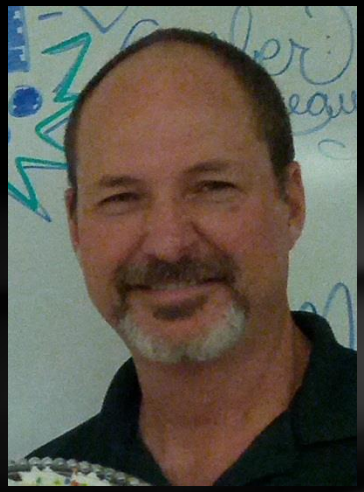
- This final method is the first one in the **`Logic-1`** section, and uses similar logic techniques as you saw in the previous method. Read the description carefully, and even click on the Show Hint button if you need to. Good luck!



CONGRATULATIONS!

- You now know how to write your own static return methods.
- You also were introduced to the AND, OR, and NOT boolean operators using the CodingBat website labs.
- *The Lesson 6A will tell you about loops and how they work.*

Thank you, and have fun!



To order supplementary materials for all the lessons in this package, such as lab solutions, quizzes, tests, and unit reviews, visit the O(N)CS Lessons website, or contact me at

John B. Owen
captainjbo@gmail.com